SCALABILITY AND PERFORMANCE ANALYSIS OF EC-EARTH 3.2.0 USING A NEW METRIC APPROACH (PART II)

BSC-CES-2016-004

EC-Earth 3.2.0, Scalability, Metrics, Computational Performance

X. Yepes-Arbós, M. C. Acosta, K. Serradell, O. Mula-Valls, F. J. Doblas-Reyes

Earth Sciences Department

Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS)

28 November 2016

TECHNICAL REPORT

Series: Earth Sciences (ES) Technical Report

A full list of ES Publications can be found on our website under:

https://earth.bsc.es/wiki/doku.php?id=library:external:technical_memoranda

Barcelona Supercomputing Center–Centro Nacional de Supercomputación (BSC-CNS)

C/Jordi Girona, 31 | 08034 Barcelona (Spain)

Summary

In this technical memorandum we present a complete scalability and performance analysis to determine the computational behavior of the EC-Earth 3.2.0 climate model. Climate science has a strong computational component, and the climate codes used in this discipline are typically complex and large in size, consuming a lot of computational resources to obtain results which can be used for scientists. This is even more important when a limited number of hours are available on a specific high performance platform. For this reason, a correct configuration avoids the waste of resources.

After performing the scalability analysis, we are able not only to identify some MPI combinations, but also to propose understandable metrics to choose the best configuration for the specific users' needs. The experiments use the T255L91 grid for IFS and the ORCA1L75 grid for NEMO. They were executed on MareNostrum III, hosted by the Barcelona Supercomputing Center (BSC). This text describes the configuration with the best speedup of EC-Earth, which is 40.3 with 15.7 SYPD, using 640 MPI processes distributed between 512 for IFS and 128 for NEMO. In contrast, the combination that uses 416 MPI processes, dedicating 288 to IFS and 128 to NEMO, has the best performance-efficiency compromise, achieving a speedup of 35 and 13.6 SYPD, but using fewer resources.

The document also describes how to estimate the number of MPI processes to achieve a desired throughput. This enables us to estimate the equivalence between the number of MPI processes and the value of simulated years per day, taking into account a good compromise between performance and efficiency.

Finally, a profiling and tracing study has been performed to determine the main bottlenecks of EC-Earth. Thus, gprof has been used for profiling and two BSC tools, Extrae and Paraver, for tracing. The results show that the model is not efficient due to large periods of synchronization. Furthermore, we focus on the coupling between IFS and NEMO through OASIS3-MCT, and we have observed that there is a large serialization due to the conservative method applied to send some fields from IFS to NEMO and runoff-mapper.

Contents

1. Introduction

Recently, there have been great improvements in the quality of climate models. One of the reasons is that computational power has grown exponentially. This leads to an enormous complexity of the models and of the experiments that could be carried out. However, it is still uncommon to find the widespread use of metrics that evaluate the performance of a model.

From a computational point of view, there is available the speedup metric, which evaluates the improvement in speed of execution time of a parallel application using different amount of resources. Although speedup is one of the most used metrics in computer science, it could not satisfy the scientists' needs. Scientists are interested in the throughput and the scalability of their models, because of the need to simulate a desired time period within a specific time to solution. For this reason, in this study we present other metrics such as the simulated years per day (SYPD) metric, which tells about the throughput of a climate model on a supercomputer, in the case of this document EC-Earth 3.2.0 on MareNostrum III, using different MPI combinations.

EC-Earth is a global coupled climate model integrating a number of component models to simulate the Earth system. The two main models are IFS 36r4 as atmospheric model and NEMO 3.6 as ocean model, both coupled using OASIS3-MCT. There are other small components like LIM3 as sea-ice model and runoff-mapper to distribute runoff from land to the ocean through rivers.

Scientists also have other concerns, such as to consume the minimum resources to perform the simulation. This requires performing slower rather than just one fast simulation. This is related to the efficiency of the model, but also to the speedup. As a consequence, a compromise between performance and efficiency is needed, which should be applied to the MPI combinations of the SYPD results. This is the idea developed in this technical memorandum.

The analysis used has been developed taking into account that the scalability analysis is bi-dimensional, setting the MPI processes used for the two model components, NEMO and IFS, at the same time. More details about our definition of a bi-dimensional search can be found in section 3.1.

Although other scalability analyses [1] have been performed in the past, one of the main goals of this document is to try to improve the information made available up to now to scientists. For example, one of the weaknesses of other studies is that the scalability results are hard to understand and show partial information. They use few MPI combinations, not representative enough, especially when considering that some combinations could be hidden in a bi-dimensional search. Furthermore, they present the quantity of IFS and NEMO processes together, making difficult to distinguish the combination of cores used for NEMO and IFS independently. On the other hand, efficiency results are not realistic as they normalize data with respect to a combination that has many processes. The base case of the efficiency plot always has value 1, which means that the number of processes used for this base case is 100% efficient, which in not always true. For a sequential program (1 process), this is always true, but using 2 or more processes, this is probably false, which is the case of EC-Earth.

Thus, if the scalability analysis of EC-Earth starts with a base case using many processes, in order to plot a value of 1 in the base case of the efficiency, all the data has to be normalized regarding the number of processes of that base case, which leads to have better efficiencies, but it is unrealistic.

On the other hand, in this document we also perform a deep performance analysis of EC-Earth using HPC tools, in this case Extrae and Paraver. Our goal is to identify the main bottlenecks of EC-Earth 3.2.0 that are limiting the throughput. We found that one of these bottlenecks is the coupling between IFS and NEMO, which is explained in detail in the document. Furthermore, it is presented the main computational problems of IFS and NEMO models independently, taking into account that some of them could be related to the coupled implementation. Individually, these issues could have a small impact on the efficiency, but the addition of their inefficiencies represents a bigger problem.

The document is organized as follows. The experimental setup, including model description, environment and model configuration, is given in the next section. The methodology used is described in section 3. Section 4 explains the results of the scalability and performance analysis. Finally, section 5 contains the conclusions of the whole study.

This study has been done using a standard resolution configuration for IFS and NEMO (T255-ORCA1). Any configuration is correct to do a performance analysis and know more details about the performance of the model. However, it would be needed to re-perform a scalability analysis for the high resolution configuration of EC-Earth 3.2.X.

2. Experimental setup

2.1. Model description

EC-Earth [2] is a global coupled climate model, which integrates a number of component models in order to simulate the earth system. It is developed by a consortium of European research institutions, which collaborate in the development of a new Earth System Model. The goal of EC-Earth is to build a fully coupled atmosphere-ocean-land-biosphere model usable for problems encompassing from seasonal-to-decadal climate prediction to climate change projections and paleoclimate simulations. It includes the following components:

- The **OASIS3-MCT coupler**: is a coupling library to be linked to the component models and which main function is to interpolate and exchange the coupling fields between them to form a coupled system.

- The **Integrated Forecasting System (IFS)** as atmosphere model: is an operational global meteorological forecasting model developed and maintained by the European Centre of Medium-Range Weather Forecasts (ECMWF). The dynamical core of IFS is hydrostatic, two-time-level, semi-implicit, semi-Lagrangian and applies spectral transforms between grid-point space and spectral space. In the vertical the model is discretized using a finite-element scheme. A reduced Gaussian grid is used in the horizontal. The IFS cycle is 36r4.

- The **Nucleus for European Modelling of the Ocean (NEMO)** as ocean model: is a state-of-the-art modelling framework for oceanographic research, operational oceanography seasonal forecast and climate studies. It discretizes the 3D Navier-Stokes equations, being a finite difference, hydrostatic, primitive equation model, with a free sea surface and a non-linear equation of state in the Jackett. The ocean general circulation model (OGCM) is OPA (Océan Parallélisé). OPA is a primitive equation model which is numerically solved in a global ocean curvilinear grid known as ORCA.

  EC-Earth 3.2.0 uses NEMO's version 3.6 with **XML Input Output Server (XIOS)** version 1.0. XIOS is an asynchronous input/output server used to minimize previous I/O problems.

- The **Louvain-la-Neuve sea-Ice Model 2/3 (LIM2/3)**: is a thermodynamic-dynamic sea-ice model directly coupled with OPA.

- The **Hydrological extension of the Tiled ECMWF Surface Scheme for Exchange processes over Land (HTESSEL)** as land and vegetation module: is part of the atmosphere model. It solves the surface energy and water balance taking into account 6 different land tiles overlying a 4 layer soil scheme. The 6 tiles are: tall vegetation, low vegetation, interception reservoir, bare soil, snow on low vegetation, and snow under high vegetation.

- The **Tracer Model 5 (TM5)** as global chemistry transport model: describes the atmospheric chemistry and transport of reactive or inert tracers.

- The **runoff-mapper** component is used to distribute the runoff from land to the ocean through rivers. It runs using its own binary and coupled through OASIS3-MCT.

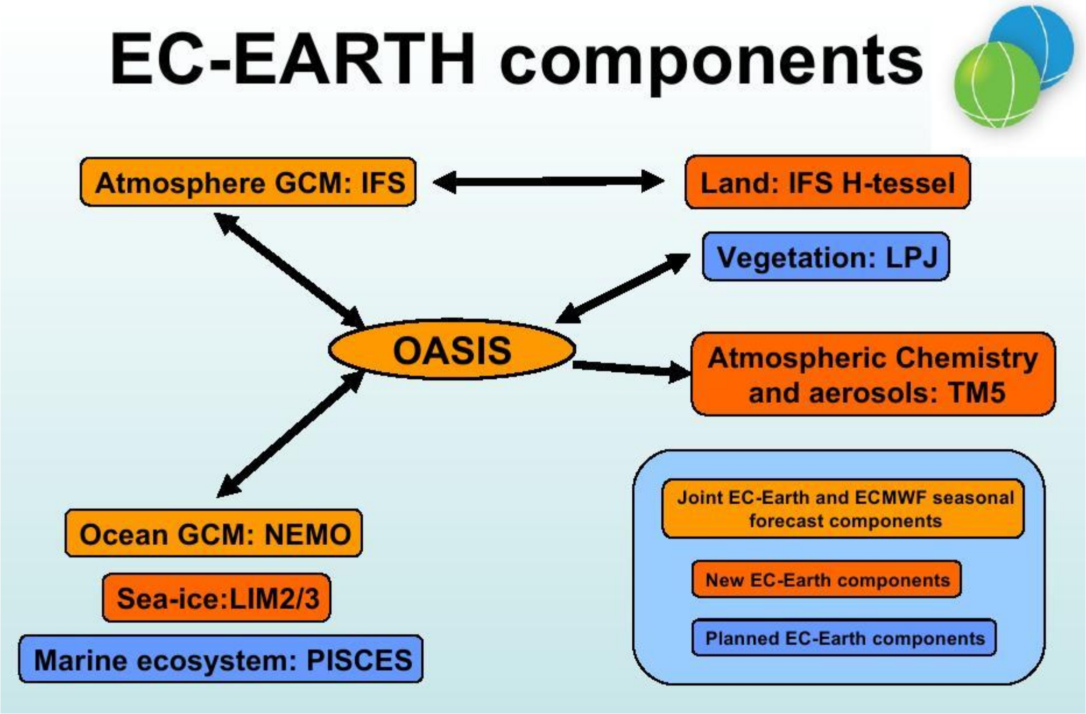


Figure 1: Components used for the EC-Earth model

The components are mainly written in Fortran, except XIOS which is written in C++. The model is parallelized using the message passing paradigm, particularly, the standard library MPI (Intel MPI v5) to be executed in distributed HPC clusters with several nodes.

For our performance analysis, we will use IFS, NEMO, LIM3, XIOS and OASIS, being these components the essential part of EC-Earth and the modules mainly used for BSC scientists in the experiments.

2.2. Environment

All analysis were done on MareNostrum III, which is a supercomputer designed by IBM and it is hosted by the Barcelona Supercomputing Center (BSC). It has the next features:

- Peak performance of 1.1 PFLOPS
- 115.5 TB of main memory
- 2 PB of disk storage
- Main nodes:
  - 2x Intel SandyBridge-EP E5-2670/1600 20M 8-core at 2.6 Ghz
  - 128 nodes with 16x8 GB DDR3-1600 DIMMS (8 GB/core)

    - 128 nodes with 16x4 GB DDR3-1600 DIMMS (4 GB/core)
    - 2752 nodes with 8x4 GB DDR3-1600 DIMMS (2 GB/core)
    - 500 GB 7200 rpm SATA II local HDD
- Operating system: Linux - SuSe distribution
- LSF queue system
- Interconnection networks:
    - Infiniband FDR10
    - Gigabit Ethernet
    - 52 racks distributed in 120 m²

2.3. Model configuration

In order to perform the analysis explained in section 3, it is necessary to set the configuration, which cannot be changed because we want to compare performance analysis. We will run the last version available, the coupled EC-Earth 3.2.0 model, so we have the following conditions:

- Revision: 2716. 2015-12-08 15:14:02 +0100 (Tue, 08 Dec 2015)
- MareNostrum III with 16 MPI processes per node. We use generic nodes with 32 GB and 16 cores, so we fill them up. The communications are done via Infiniband and the I/O is done via a Gigabit Ethernet network.
- 1 chunk of 3 months (to minimize initialization's overhead)
- Average of 3 identical executions
- Without check flags
- Default namelists values
- Optimum mpirun order: in order to exploit better processor affinity. Order: IFS, NEMO, Runoff-mapper and XIOS.
- Use of an optimization to avoid mpi_allgather use a the northfold
- Fortran compilation flags: -O2 -r8 –xHost
- C compilation flags: -O2 –xHost
- Compiler: Intel v13.0.1
- Intel MPI library v5.0.1.035
- MKL library v11.1.2
- NetCDF library v4.3.2
- HDF5 library v1.8.12-mpi
- GRIB library v.1.14.0

Furthermore, we have the following specific parameters:

Model	Nemo-3.6
Configuration name	ORCA1L75_LIM3
Resolution	ORCA1L75
Modules	OPA and LIM3
Time step	2700 seconds (32 time steps per day)
Compilation keys	key_trabbl key_vvl key_dynspg_ts key_ldfslp key_traldf_c2d key_traldf_eiv key_dynldf_c3d key_zdfddm key_zdftmx key_mpp_mpi key_zdftke key_lim3 key_iomput key_oasis3 key_oa3mct_v3

Table 1: Configuration parameters for NEMO

Model	ifs-36r4
Resolution	T255L91
Time step	2700 seconds (32 time steps per day)

Table 2: Configuration parameters for IFS

Component model	OASIS3-MCT
Coupling frequency	2700 seconds (default value)

Table 3: Configuration parameters for OASIS3-MCT

On the other hand, we used Autosubmit to automate the scalability analysis. It is a python-based tool to create, manage and monitor experiments by using computing clusters, HPCs and supercomputers remotely via ssh. It has support for experiments running in more than one HPC and for different workflow configurations.

3.Methodology

In this section we will go through subsections that explain which is the methodology used to perform the scalability analysis, how we do the profiling and which are the tools used to trace the model.

3.1. Scalability

We do the scalability analysis with the following goals:

- How EC-Earth 3.2.0 scales, and thus, its performance.
- Which are the ideal combinations of MPI processes taking into account different points of view, i.e., find out which is the fastest combination, or also find out a combination with a good performance-efficiency compromise according to our requirements. In order to do this, the results present not only a computational and scientific perspective, but also a new one, achieving a compromise between both of them, easy to understand by anyone.
- How many MPI processes have to be set to achieve a desired throughput. This means that the reader will be able to easily know the number of MPI processes to use if he or she wants to achieve a particular value for speedup, efficiency or simulated years per day. It will be also possible to tune this combination taking into account the performance-efficiency compromise, i.e., if the reader prefers more performance or more efficiency.

It is important to mention that the scalability analysis of EC-Earth might be one-dimensional, i.e., obtaining the best number of processes for IFS and NEMO independently. This is because it is not clear if IFS and NEMO models are dependent between them in terms of execution time. This means that if the number of processes of one model is fixed and we only iterate over the number of processes of the other model, the execution time of the model with fixed processes should not be affected and have always the same value. Coupling profiling tools such as LUCIA will be used for successive tests in order to clarify this and determine if IFS and NEMO models are dependent or not.

We present a methodology that is suitable for scalability analyses of dependent models (bi-dimensional search), but it is still valid for independent models (one-dimensional search). The difference between a one-dimensional and a bi-dimensional analysis is the amount of tests to be performed, being larger in the bi-dimensional.

However, taking into account the third goal presented above, in this study is necessary to use the bi-dimensional search, as we want to make easy the interpretation of the speedup, efficiency and simulated years per day charts. Otherwise, using the one-dimensional search would not be possible.

So using a bi-dimensional search, we programmed a script that handles all the work of the scalability analysis. Figure 2 shows the flowchart that follows our script. This script sets different number of NEMO and IFS processes for several MPI combinations automatically. Besides IFS and NEMO processes, we also have to set one process for XIOS and one process for Runoff-mapper.

The criterion used to choose the MPI combinations of IFS and NEMO processes is based in powers of two: 1, 2, 4, 8, 16, 32, 64, 128, 256 and 512 for both IFS and NEMO, except 1, which is used only by IFS. NEMO is executed from 2 processes because we were not able to run it in coupled mode with 1 MPI process for these tests.

However, this is an exponential growth, so the distance between numbers becomes too large. These holes are filled up with multiples of two: 192, 224, 288, 320 and 384 for both IFS and NEMO, and 640, 768 and 896 only for IFS.

For each iteration of the MPI combinations, we create a new experiment with three identical members that are used to get the average time of three identical executions. Furthermore, we store each experiment ID in a file, which is used in the post-processing script. But we can also use the IDs' file to re-perform the scalability analysis in a future without creating all the experiments again.

The post-processing script reads the three execution times of each MPI combination, does the average and stores it again.

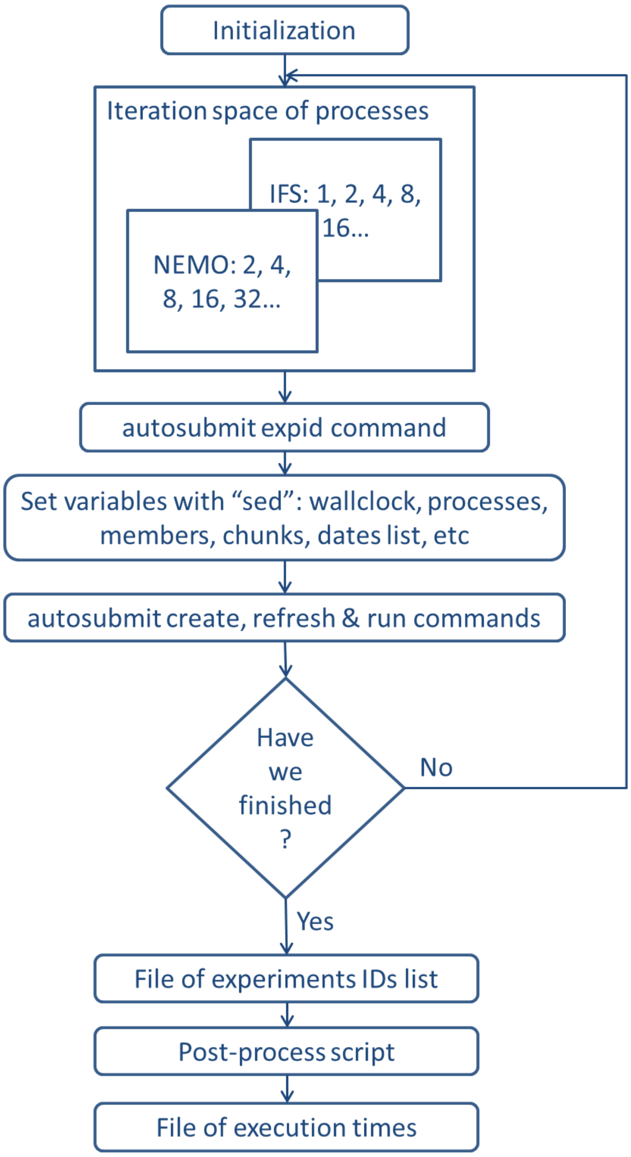

Initialization
Iteration space of processes
IFS: 1, 2, 4, 8, 16…
NEMO: 2, 4, 8, 16, 32…
autosubmit expid command
Set variables with "sed": wallclock, processes, members, chunks, dates list, etc
autosubmit create, refresh & run commands
Have we finished ?
No
Yes
File of experiments IDs list
Post-process script
File of execution times

Figure 2: Flowchart of the scalability analysis methodology

Using the execution time and the number of MPI processes, we can calculate the speedup, the efficiency and the simulated years per day. The formulas are:

$$Speedup = \frac{Execution\ time\ base\ combination}{Execution\ time}$$

Where:

- $Execution\ time\ base\ combination$ is the execution time of the combination that uses fewer MPI processes.
- $Execution\ time$ is the execution time of the rest of combinations that use more MPI processes than the base combination.

$$Efficiency = \frac{Speedup}{\left(\frac{\#MPI\ processes}{\#MPI\ processes\ base\ combination}\right)}$$

Where:

- $\#MPI\ processes\ base\ combination$ is the number of MPI processes of the combination that uses fewer MPI processes. It is used to normalize the data when the base case uses 2 or more processes, which is the case of EC-Earth.
- $\#MPI\ processes$ is the number of MPI processes used by a combination to get its $Speedup$.

$$SYPD = \left(\frac{3\ \cancel{months}}{Execution\ time\ \cancel{(s)}}\right)\left(\frac{1\ year}{12\ \cancel{months}}\right)\left(\frac{3600\ \cancel{s}}{1\ \cancel{h}}\right)\left(\frac{24\ \cancel{h}}{1\ day}\right) = \frac{21600}{Execution\ time\ (s)}{}^{Years}\!/_{Day}$$

Where:

- $Execution\ time\ (s)$ is the execution time of a combination expressed in seconds.

Since the scalability analysis is bi-dimensional and one of the goals is to provide an easy and understandable representation of the data, we have to define a process of selecting it accurately.

To do so, we decided to reduce the amount of data to show by selecting the most relevant and important one. Our methodology was to introduce a new metric, which gives a value to each MPI combination. This metric evaluates the performance-efficiency compromise in order to represent and study only the most relevant IFS-NEMO combinations. It is calculated as follows:

$$Performance\text{-}Efficiency\ Compromise = Speedup \times Efficiency$$

This formula penalizes combinations that have a low efficiency, but benefits the ones that have a good efficiency. It is used to get a two dimensional mask that we apply later to the common metrics: execution time, speedup, efficiency and simulated years per day. With this mask we can obtain 2D charts of the listed metrics.

The procedure used to get this mask is based on creating subsets of MPI processes, first for IFS and then for NEMO. This means to fix IFS processes and iterate over all NEMO processes. To clarify it, it is done as follows:

- Subset 1.1: IFS = 1 and NEMO = 2, 4, 8, 16…
- Subset 1.2: IFS = 2 and NEMO = 2, 4, 8, 16…
- Subset 1.3: IFS = 4 and NEMO = 2, 4, 8, 16…
- Subset 1.4: IFS = 8 and NEMO = 2, 4, 8, 16…
- …

Then, for each 1.x subset we take the best value. We do the same process by fixing NEMO processes and iterating over all IFS processes:

- Subset 2.1: NEMO = 2 and IFS = 1, 2, 4, 8…
- Subset 2.2: NEMO = 4 and IFS = 1, 2, 4, 8…
- Subset 2.3: NEMO = 8 and IFS = 1, 2, 4, 8…
- Subset 2.4: NEMO = 16 and IFS = 1, 2, 4, 8…
- …

Again, we take the best value for each 2.x subset. This gives a mask of MPI combinations that have the best performance-efficiency compromise. The mask has combinations over all processes range, i.e., from 1 to 896 in IFS and 2 to 512 in NEMO. Otherwise, if we had not used the subset methodology, we only would have got a subrange of combinations (the ones with higher values).

On the other hand, once we have the mask, we have to apply it into the common metrics that we listed above. Using this, we get a list with the relevant data instead of the original table with all the data. Then, we sort this list by ascending order of total MPI processes (IFS + NEMO). At this point, we are able to represent the data in a 2 dimensional chart.

3.2. Profiling

Profiling is the analysis of the application's behavior using information gathered as the program executes in order to determine which parts need to be optimized. To achieve this goal, a profiling tool takes into account aspects such as execution time per subroutine, execution time per line of code (this implies to know which instructions have more cost), which are the dependencies between functions (it builds a tree with calls), the number of times that each function has been called, etc. We can get all this data without looking at the source code or even without having to instrument it.

In our case we used gprof, which is supported by the compiler. In order to use it, we have to compile with flags -g and -pg without modifying or instrumenting the source code. When it is executed (we simulated one chunk of 6 months and using 512 MPI processes for IFS and 128 for NEMO), it generates as many gmon.out files as MPI processes that contain the profile of each process. Then, we have to separate the gmon.out files according to the component which belong. This means three groups: NEMO, IFS and runoff-mapper. For each group, using gprof again, we merged them in order to have only one gmon.out.

Finally, we can apply a python script to the three gmon.out files to generate the dependency graphs.

3.3. Tracing

In addition to the profiling analysis, we also perform a complete performance analysis of EC-Earth by tracing it. Tracing is the process of recording event-based performance data along the execution of a program.

The tools used to trace the model are Extrae and Paraver, which are open-source and developed by the BSC Performance Tools group.

Extrae is a package used to instrument the code automatically and/or manually through its API. It generates Paraver trace-files with hardware counters, MPI messages and other information for a post-mortem analysis.

Paraver is a trace browser that can show a global visual qualitative perspective of the application behavior for later focus on the bottlenecks with a detailed quantitative analysis.

4. Results

In this section we will go through subsections that explain how EC-Earth 3.2.0 scales and where are the performance bottlenecks.

4.1. Scalability

In the first part, we only present all the data with 3D charts. Then, we go into details with 3 subsections that analyze the results more accurately.

Note that the 3D charts are rotated to maximize the visualization of the data, so IFS and NEMO processes axes are not always in the same position and/or direction. Figures 3, 4, 5, 6 and 7 show the execution time, speedup, efficiency, simulated years per day and performance-efficiency compromise respectively.

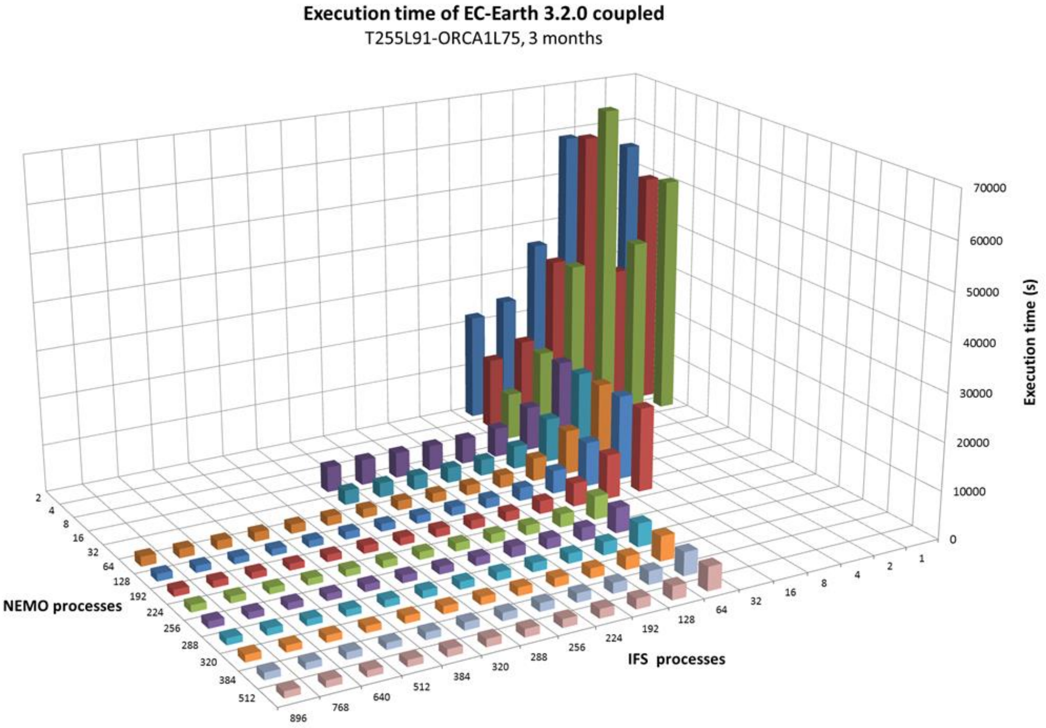


Figure 3: Execution time of EC-Earth 3.2.0 coupled in a 3D chart

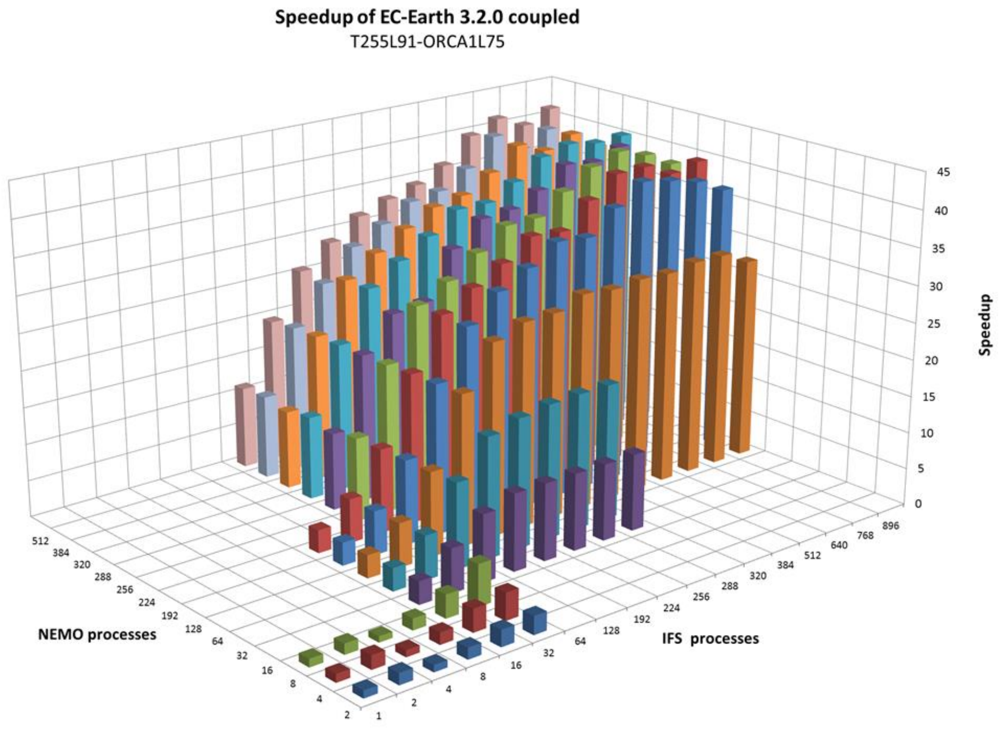


Figure 4: Speedup of EC-Earth 3.2.0 coupled in a 3D chart

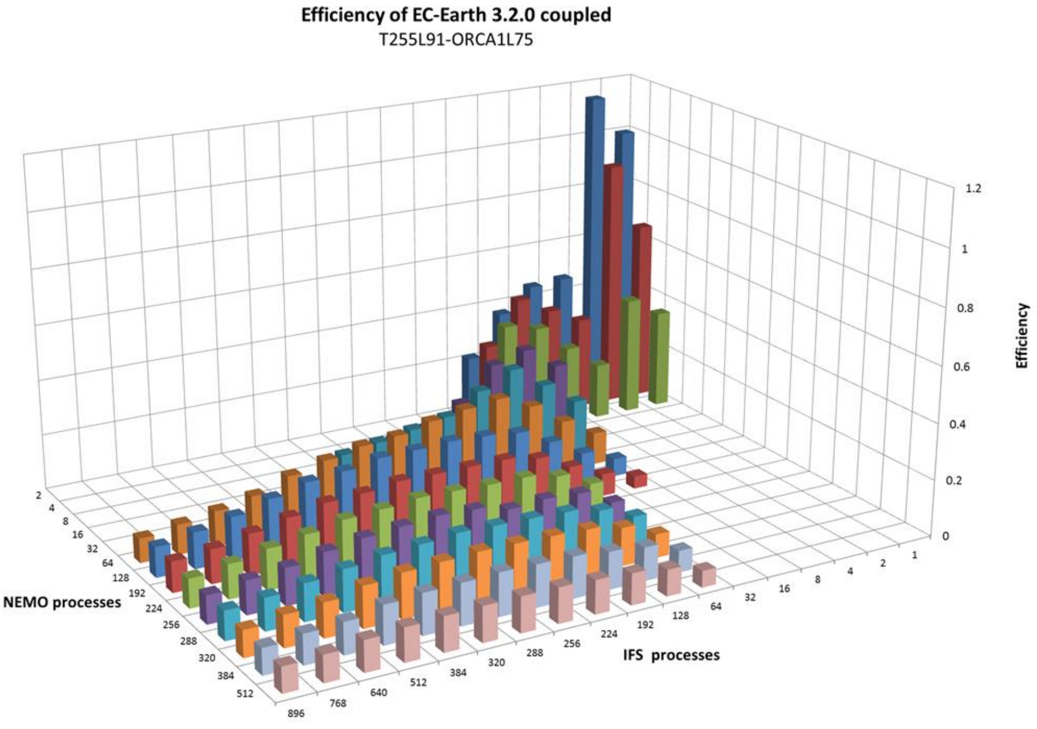


Figure 5: Efficiency of EC-Earth 3.2.0 coupled in a 3D chart

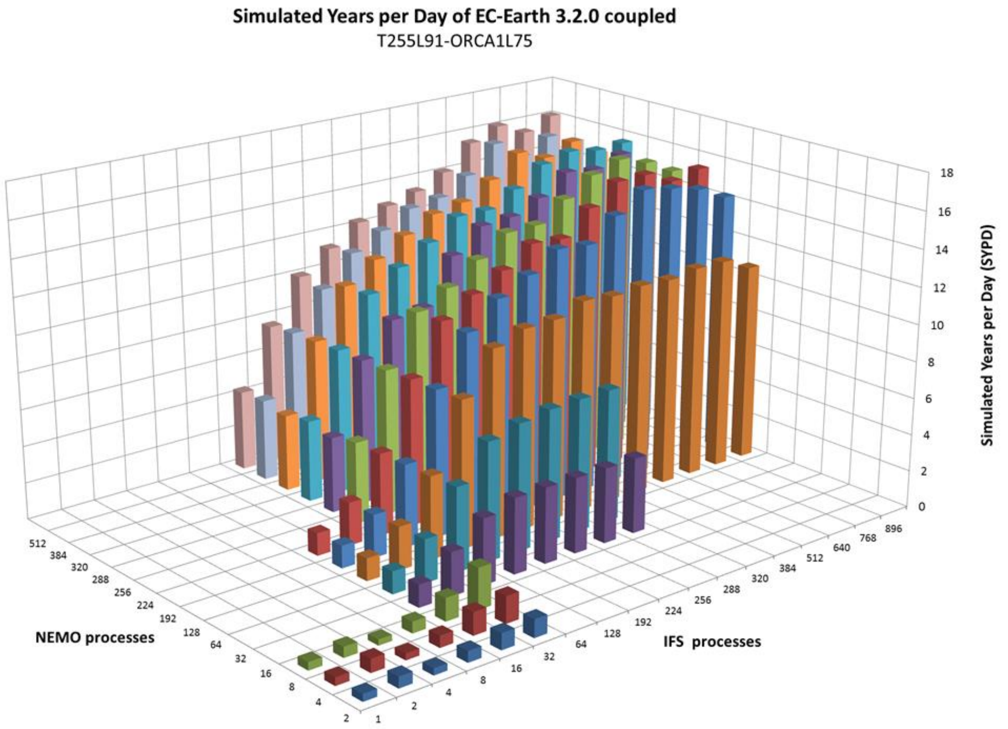


Figure 6: Simulated Years per Day of EC-Earth 3.2.0 coupled in a 3D chart

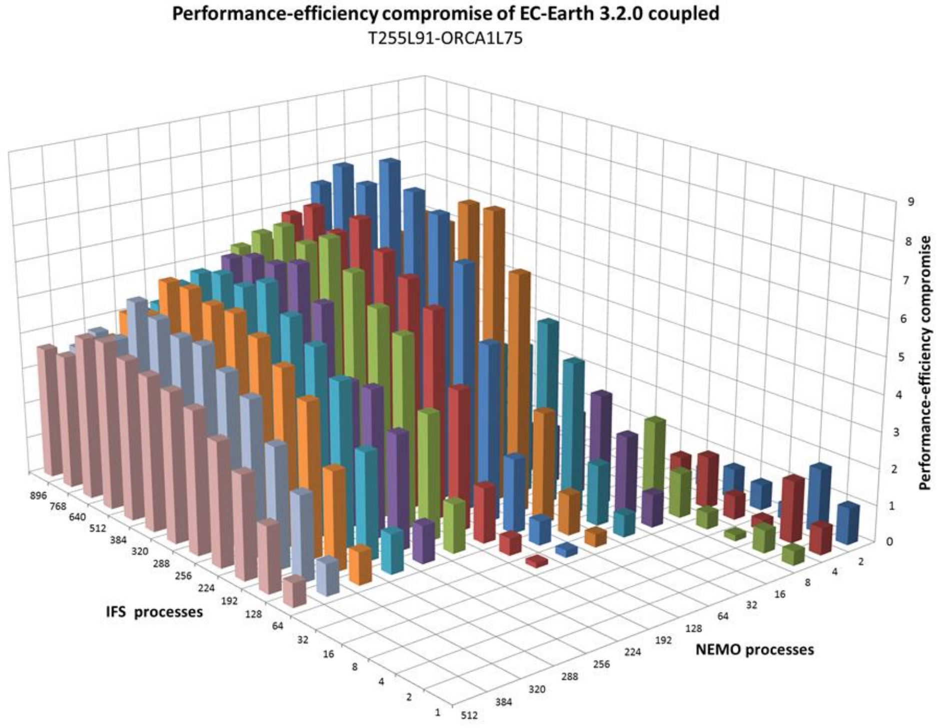


Figure 7: Performance-efficiency compromise of EC-Earth 3.2.0 coupled in a 3D chart

These charts show that EC-Earth 3.2.0 does not scale well, because the efficiency is really low. But clearly, interpreting these 3D charts is hard, so the following charts are in 2D to make the analysis easier.

4.1.1. Computational perspective

In this subsection we will present how EC-Earth 3.2.0 scales and we will identify which is the fastest combination and the more efficient. Note that to express the number of processes, we use this notation: "Total number of processes [IFS+NEMO]", where the two values within the brackets express how the "Total number of processes" are distributed between IFS and NEMO.

Applying the methodology explained in section 3.1, we make the charts of Figures 8, 9 and 10 which show execution time, speedup and efficiency respectively. We can see again and more clearly that EC-Earth 3.2.0 does not scale well at all. The efficiency is truly bad, as it plummets. There is an exception with the 4 [2+2] combination, which gives a super-linear speedup (efficiency above 1), but it is not useful because the speedup is too small. This super-linear speedup could be related to memory issues, like available bandwidth per process, amount of memory available per process, memory accesses collisions, etc.

The reason why EC-Earth 3.2.0 does not scale well is because using more processes implies higher fine-grained decomposition and higher overhead of the MPI communications, and this leads to important imbalances. This also explains the low efficiency.

Despite the bad performance, the speedup chart also shows that it increases relatively constant up to 416 [288+128] processes and, at most, up to 640 [512+128] processes. At this point, it becomes more or less flat.

On the other hand, the fastest combinations are 640 [512+128] and 896[768+128], but taking into account the number of processes, and as a consequence the efficiency, it is only worth to point out the 640 [512+128] combination as the fastest.

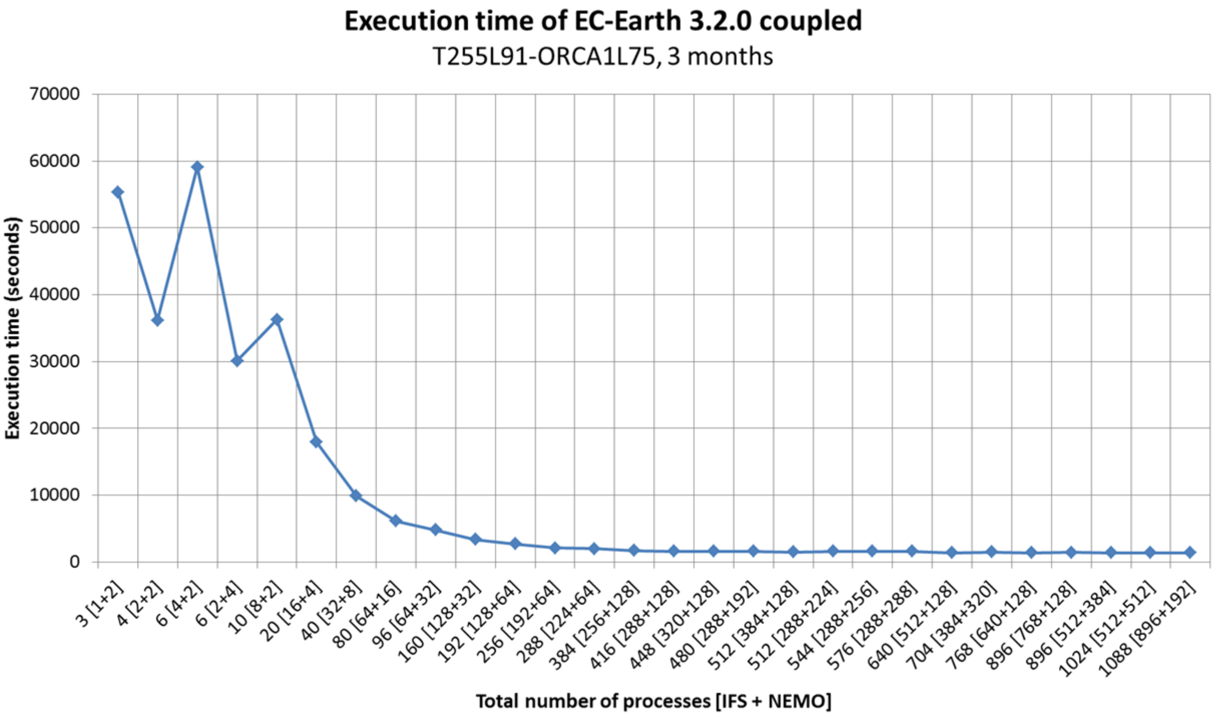


Figure 8: Execution time of EC-Earth 3.2.0 coupled in a 2D chart

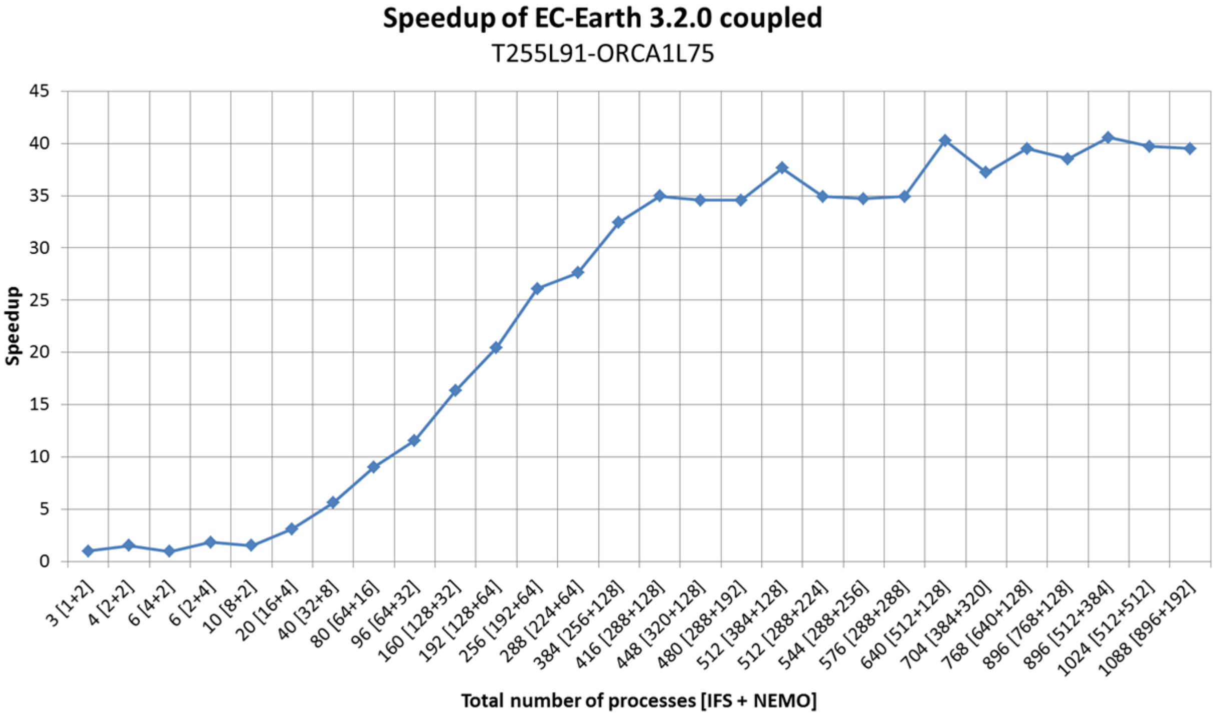


Figure 9: Speedup of EC-Earth 3.2.0 coupled in a 2D chart

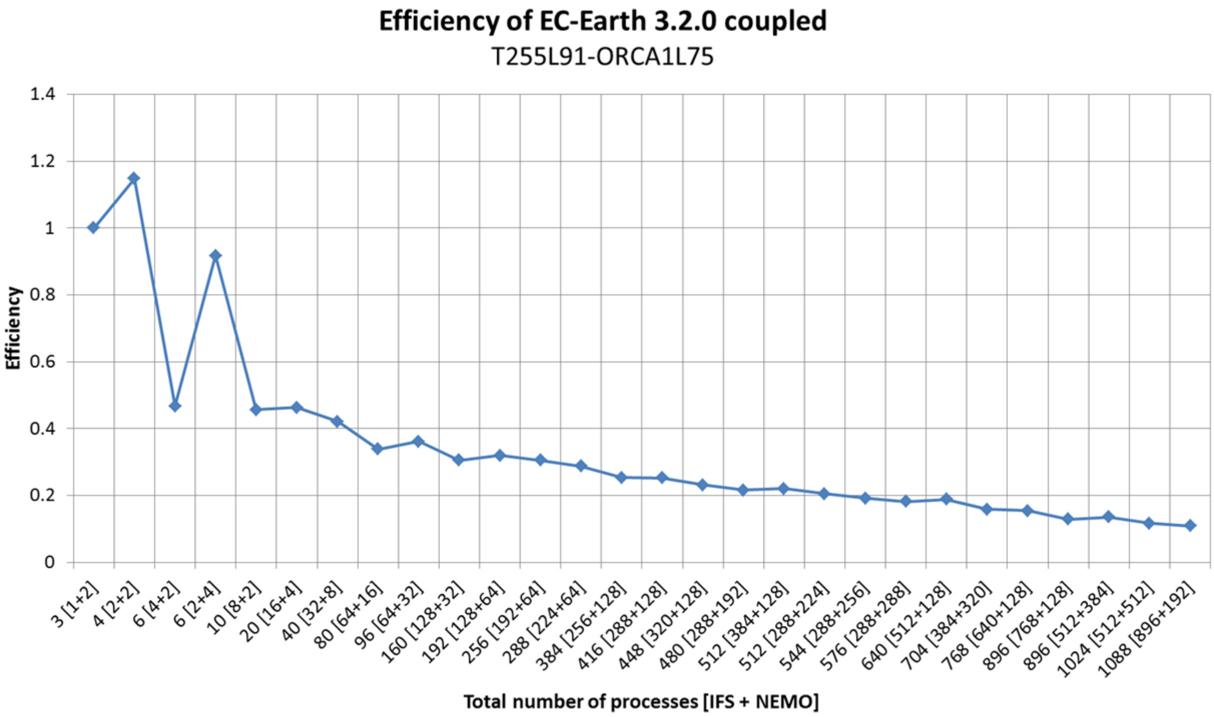


Figure 10: Efficiency of EC-Earth 3.2.0 coupled in a 2D chart

4.1.2. Scientific perspective

It is useful to choose the amount of MPI processes needed to achieve a desired throughput. For this purpose, the scientific community uses the simulated years per day metric, which tells you how many years can be simulated in a day of real life. Figure 11 shows this metric. Ideally, to perform an execution at a desired rate, one would select the SYPD on the vertical axis and read off the MPI combination that achieves it.

The speedup and SYPD charts have the same shape. This is because they are in function of the execution time and a constant value. As we saw in Figure 9, there is a steady increase up to 416 [288+128] processes in the SYPD chart, and, at most, we can get about 15.7 SYPD with 640 [512+128] and 896 [768+128] combinations. But taking into account the efficiency, it is only worth to use 640 [512+128] processes. We remark that it is not possible to get a throughput higher than 16 SYPD, since the speedup does not go beyond about 40.

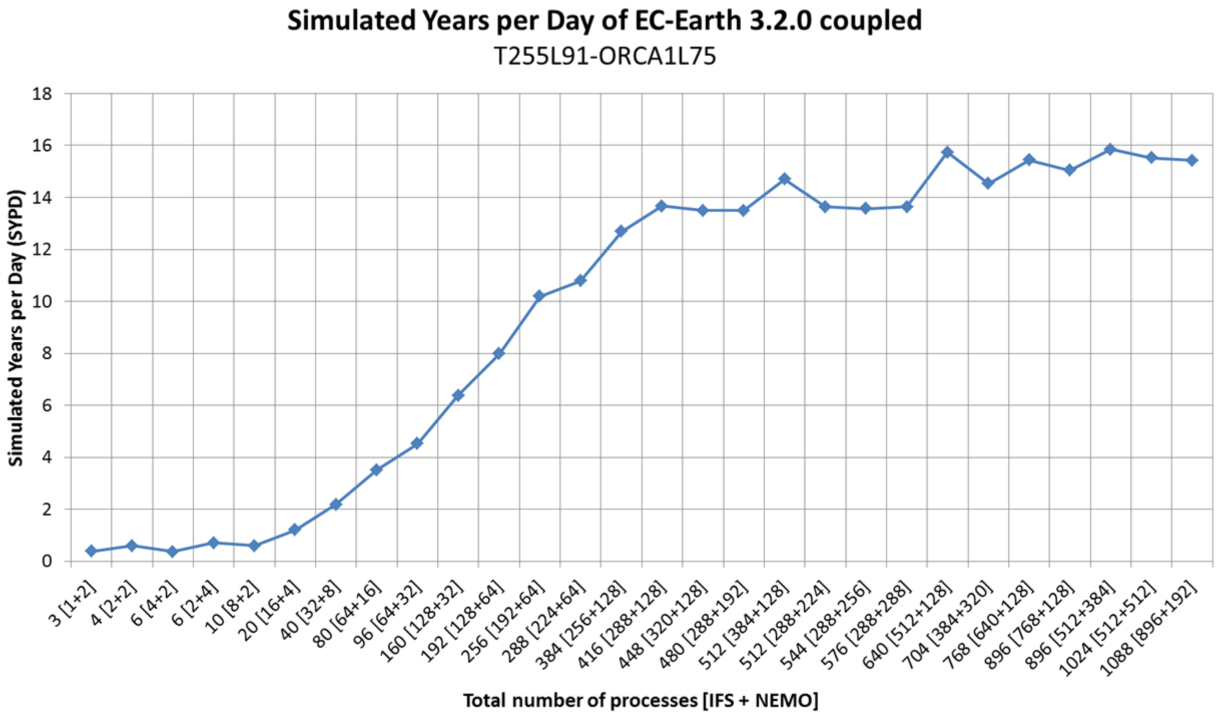


Figure 11: Simulated Years per Day of EC-Earth 3.2.0 coupled in a 2D chart

4.1.3. Performance-efficiency compromise metric

As we explained in section 3, representing 3D data in a 2D chart is quite challenging. To do it, we introduced the metric performance-efficiency compromise which gives a value to each MPI combination. Then applying the technique of selecting the best of each subset, we obtain the mask that we apply later to the common metrics.

Table 4 shows the scores of each MPI combination. There are highlighted the best values in both 1.x (blue) and 2.x (orange) subsets. When the best value coincides in both subsets, it is highlighted in brown. Red and green are used to highlight the worst and the best values respectively.

Perf.-eff. compromise		NEMO processes													
		2	4	8	16	32	64	128	192	224	256	288	320	384	512
IFS processes	1	1	0.7388566	0.39206962											
	2	1.75502733	1.68400603	0.62326996											
	4	0.4374124	0.30922948	0.16308627											
	8	0.69606613	0.63721224	0.45201601											
	16	0.82681647	1.42985835	1.22275921	0.89296316	0.59878015	0.36023554	0.1997723	0.13925388						
	32	0.53793904	1.1179155	2.36018977	2.19958278	1.65679091	1.10452727	0.65564668	0.47184536						
	64				3.0490711	4.17841384	3.05030581	2.03774003	1.51326336	1.34236234	1.03640388	1.09200895	0.88816044	0.86401138	0.64966174
	128				2.24299986	4.98950028	6.53138926	4.85535149	3.86717612	3.46957582	3.17349144	2.95085301	2.70827448	2.339009	1.800255
	192				1.55810548	4.0940055	7.98414277	6.75861176	5.74320734	5.27551738	4.07892876	4.53211619	4.23235051	3.30090493	2.82805647
	224				1.34378147	3.84513106	7.93534449	7.84658642	6.32999784	5.74775644	3.97555562	5.16524418	4.83540256	4.24868756	3.38645377
	256				1.18302556	3.48609135	7.19920256	8.23302911	6.80681537	6.45686996	5.81428035	5.6972382	5.35578537	4.66736746	3.91176847
	288				1.0676774	3.14580445	7.26135314	8.817858	7.46172035	7.13739703	6.64186004	6.34435292	5.75337782	5.11030468	4.11450421
	320						6.46371472	7.99470038	6.86602147	6.75160611	6.39269067	5.99529042	5.70991762	5.05588793	4.24098533
	384						5.69331865	8.29863415	7.34652681	7.00739122	6.35038904	6.08845776	5.8980696	5.2849513	4.40020058
	512						4.36345954	7.59974679	6.92903257	6.5892648	6.09991133	5.88111061	5.83745277	5.50833633	4.62106145
	640						3.70858587	6.09991133	5.88877079	5.98133297	5.10370607	5.32620625	4.68373724	4.21794693	4.45917766
	768						3.1280112	4.96253156	4.66411952	4.88880937	4.77763886	4.53298282	4.46393447	4.16124544	3.68624888
	896						2.38060335	3.91149121	4.29967079	3.89910901	3.78557354	4.09602947	3.49892381	3.46411602	3.63751916
		🟥 MIN. SCORE			🟩 MAX. SCORE				🟧	+	🟦	=	🟫		

Table 4: Performance-efficiency compromise table with the best values of the subsets 1.x (blue) and 2.x (orange) highlighted. In red and in green are highlighted the worst and the best values respectively.

Finally, if we take all the highlighted combinations and we sort them by ascending order of the total MPI processes, we get the performance-efficiency compromise chart of Figure 12.

From this chart we can draw some points:

- The 416 [288+128] combination has the best performance-efficiency compromise.
- The 6 [4+2] combination has the worst performance-efficiency compromise, which is also the slowest in terms of execution time.
- In section 4.1.1 we pointed out 640 [512+128] and 896[768+128] combinations as the fastest ones, but it was only worth to take into account the 640 [512+128] one. With this chart we can see that we were right, because if we compare their values, clearly the 640 [512+128] combination has better performance-efficiency compromise.

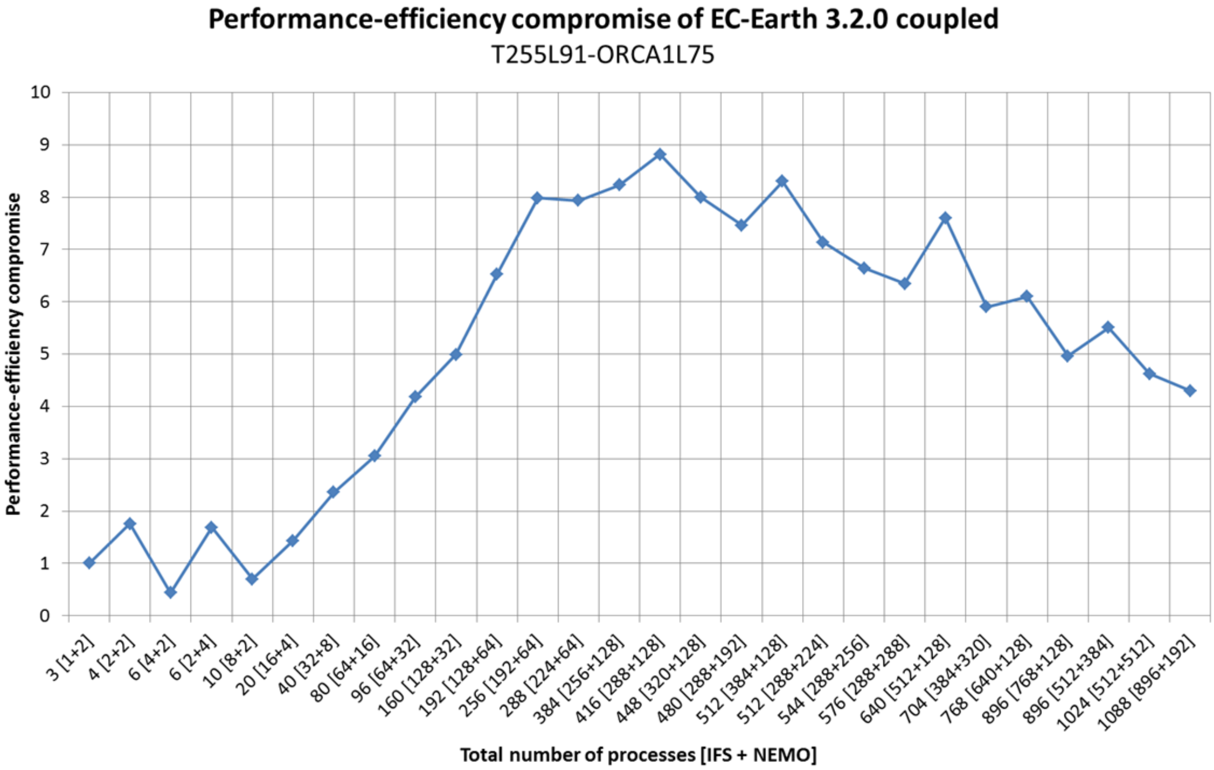


Figure 12: Performance-efficiency compromise of EC-Earth 3.2.0 coupled in a 2D chart

On the other hand, if one is mainly concerned about the efficient usage of resources, this metric reflects that it is better to fix an upper limit, up to 416 [288+128] processes, and also 512 [384+128] and 640 [512+128]. It is important to understand that when we increase the performance (or throughput), we lose efficiency. On the other hand, if we need the simulations to finish in a reasonable time, it is necessary to put a lower limit, around 256 [192+64] processes (also keeping the previous upper limit).

This chart also can be used to compare combinations with the simulated years per day chart. If an EC-Earth user wants to know the combination that achieves a desired throughput, but there are 2 or more combinations that achieve it, he or she can compare these combinations using the performance-efficiency compromise chart and use the one with the best value. For example, there are 6 combinations that give around 13.6 SYPD: 416 [288+128], 448 [320+128], 480 [288+192], 512 [288+224], 544 [288+256] and 576 [288+288]. But we should choose 416 [288+128] processes because it is the one with the highest performance-efficiency compromise. Obviously, it will be always the combination that uses fewer processes.

4.2. Profiling

In section 3.2 we explained that we used gprof to profile EC-Earth because it is very easy to use and results are shown in a visual way.

Figures 13, 14, and 15 show the dependency graphs of IFS, NEMO and runoff-mapper respectively. The values in the graphs cannot be appreciated because their size is constrained by the size of a DIN A4 sheet. However, the original graphs can be checked on the provided URL of the footnote[1].

[1] The original dependency graphs are available here: https://goo.gl/z9MZCF

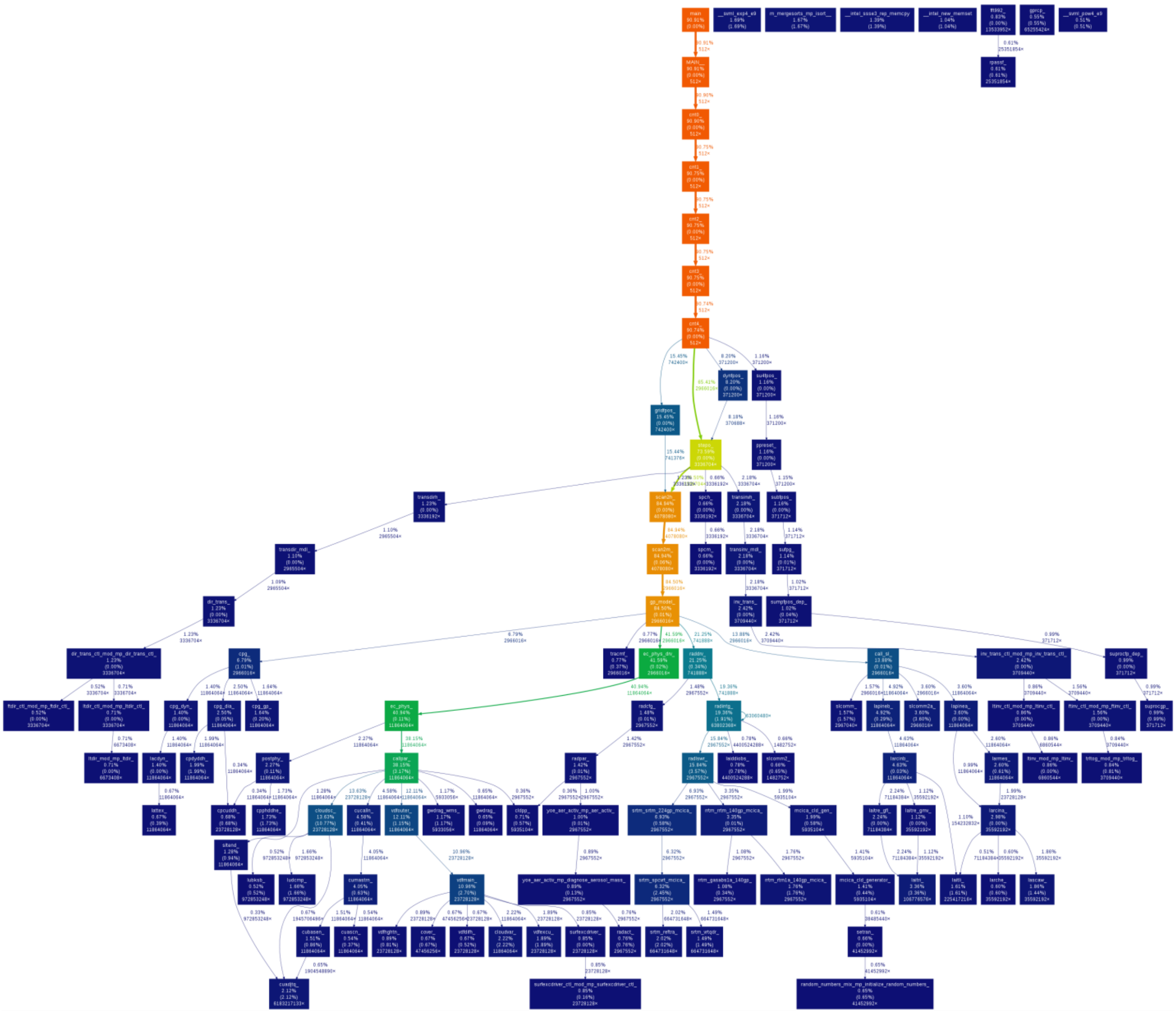


Figure 13: Dependency graph of IFS' functions

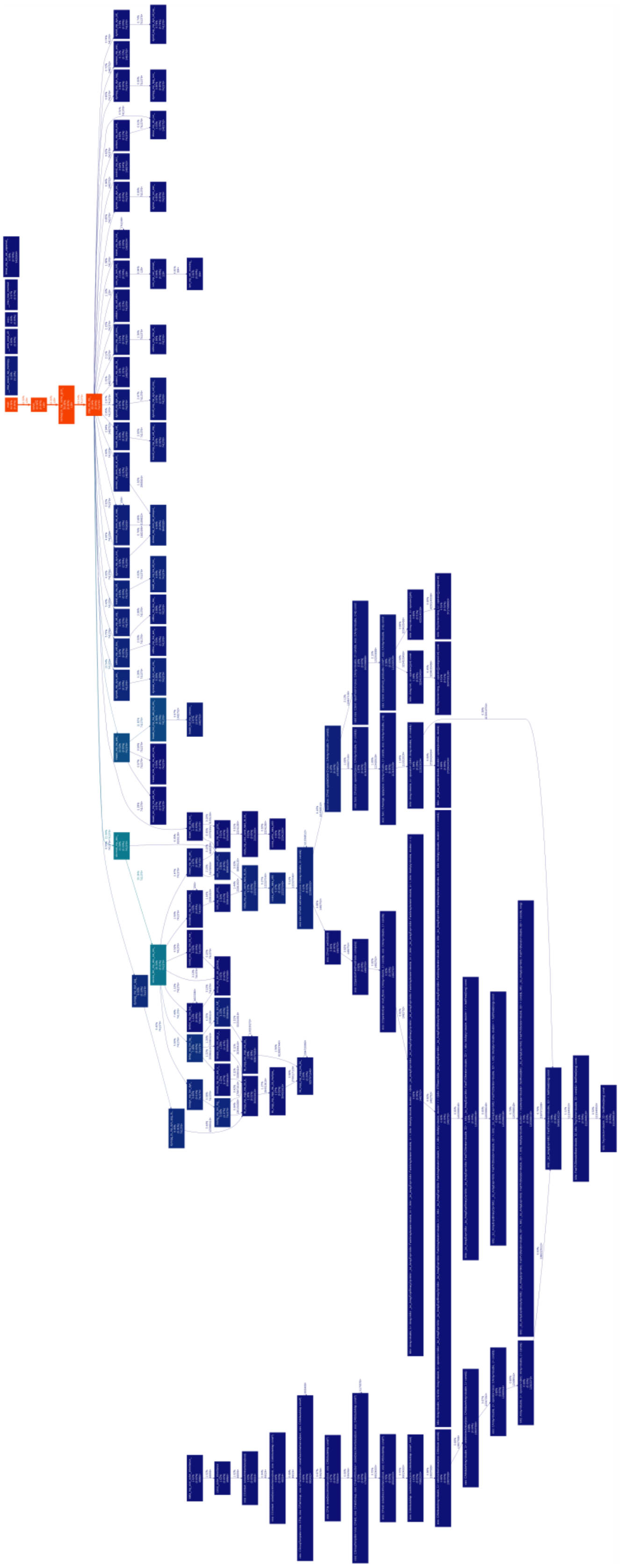


Figure 14: Dependency graph of NEMO's functions

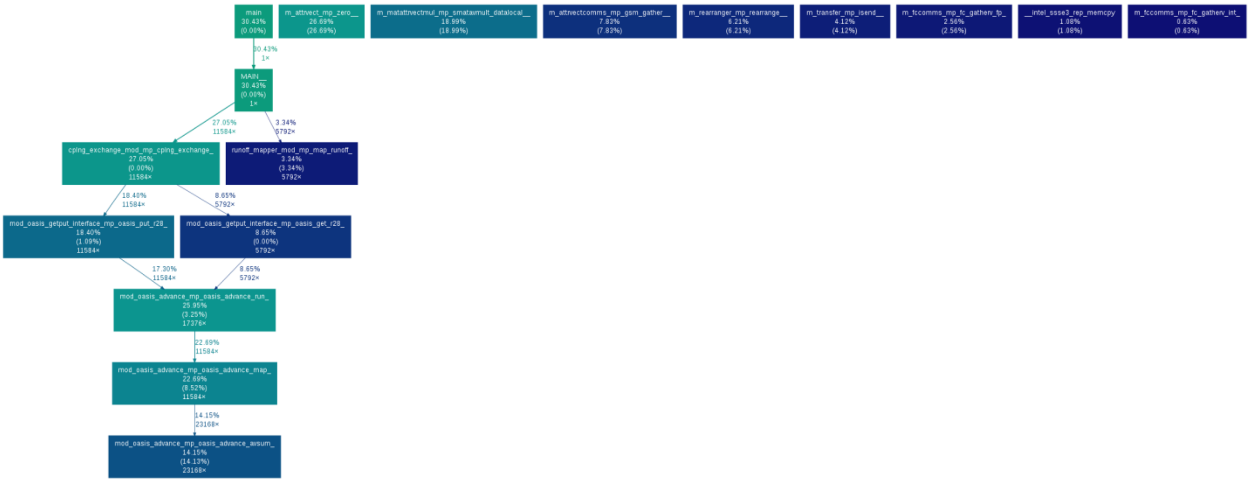

main
30.43%
(0.00%)
m_attrvect_mp_zero__
26.69%
(26.69%)
m_matattrvectmul_mp_smatavmult_datalocal__
18.99%
(18.99%)
m_attrvectcomms_mp_gsm_gather__
7.83%
(7.83%)
m_rearranger_mp_rearrange__
6.21%
(6.21%)
m_transfer_mp_isend__
4.12%
(4.12%)
m_fccomms_mp_fc_gatherv_fp_
2.56%
(2.56%)
__intel_ssse3_rep_memcpy
1.08%
(1.08%)
m_fccomms_mp_fc_gatherv_int_
0.63%
(0.63%)
30.43%
1×
MAIN__
30.43%
(0.00%)
1×
27.05%
11584×
3.34%
5792×
cplng_exchange_mod_mp_cplng_exchange_
27.05%
(0.00%)
11584×
runoff_mapper_mod_mp_map_runoff_
3.34%
(3.34%)
5792×
18.40%
11584×
8.65%
5792×
mod_oasis_getput_interface_mp_oasis_put_r28_
18.40%
(1.09%)
11584×
mod_oasis_getput_interface_mp_oasis_get_r28_
8.65%
(0.00%)
5792×
17.30%
11584×
8.65%
5792×
mod_oasis_advance_mp_oasis_advance_run_
25.95%
(3.25%)
17376×
22.69%
11584×
mod_oasis_advance_mp_oasis_advance_map_
22.69%
(8.52%)
11584×
14.15%
23168×
mod_oasis_advance_mp_oasis_advance_avsum_
14.15%
(14.13%)
23168×

Figure 15: Dependency graph of runoff-mapper's functions

These graphs are useful to know how the components are organized, building an explicit hierarchy of the source code. Additionally, they can be used to place the main bottlenecks found in the tracing part of this study.

To know additional details, the next step in a profiling analysis consists in highlighting the functions with the largest computational cost. Therefore, graphs of the previous components are summarized and only relevant subroutines are presented. Figures 16, 17 and 18 show the execution time in percentage of the most computationally expensive functions of IFS, NEMO and runoff-mapper components respectively.

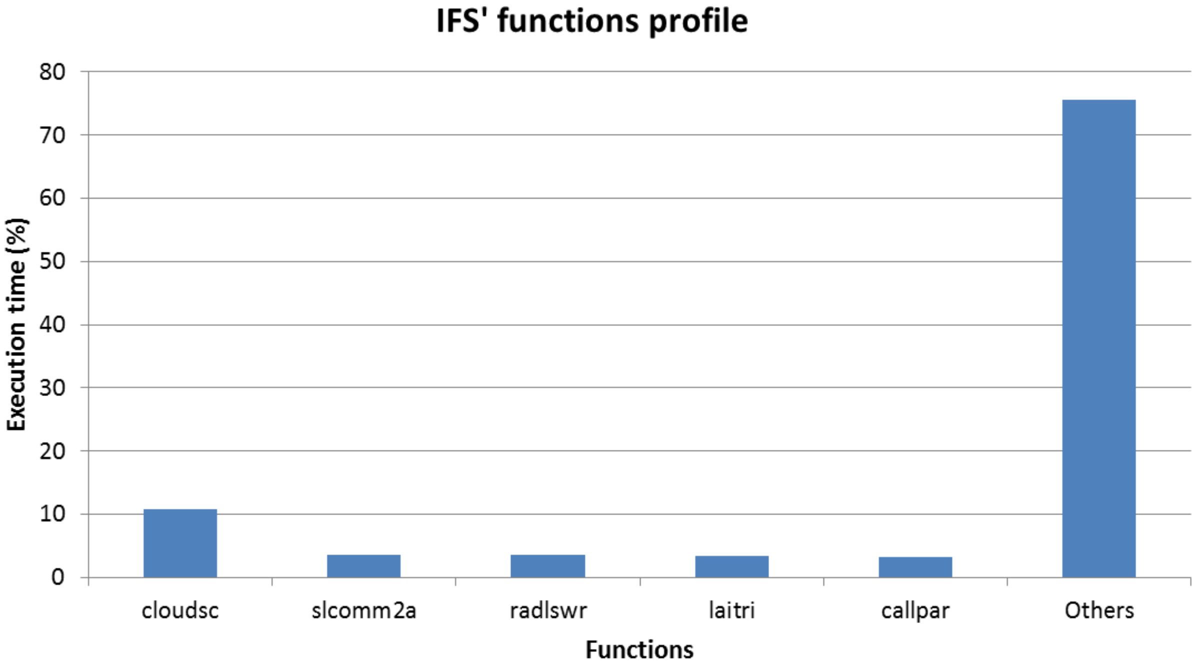


Figure 16: Profile of the execution time of the largest computational cost of IFS' functions

Cloudsc function takes around 10% of the execution time, indicating that this subroutine could be a starting point to optimize IFS. There are other functions such as *slcomm2a* or *radlswr* that could be interesting to study more in detail. However, note that the execution time is very distributed among many functions (*Others* represent more than a 75% of the total time), and optimizations for specific functions do not seem the best strategy in this case.

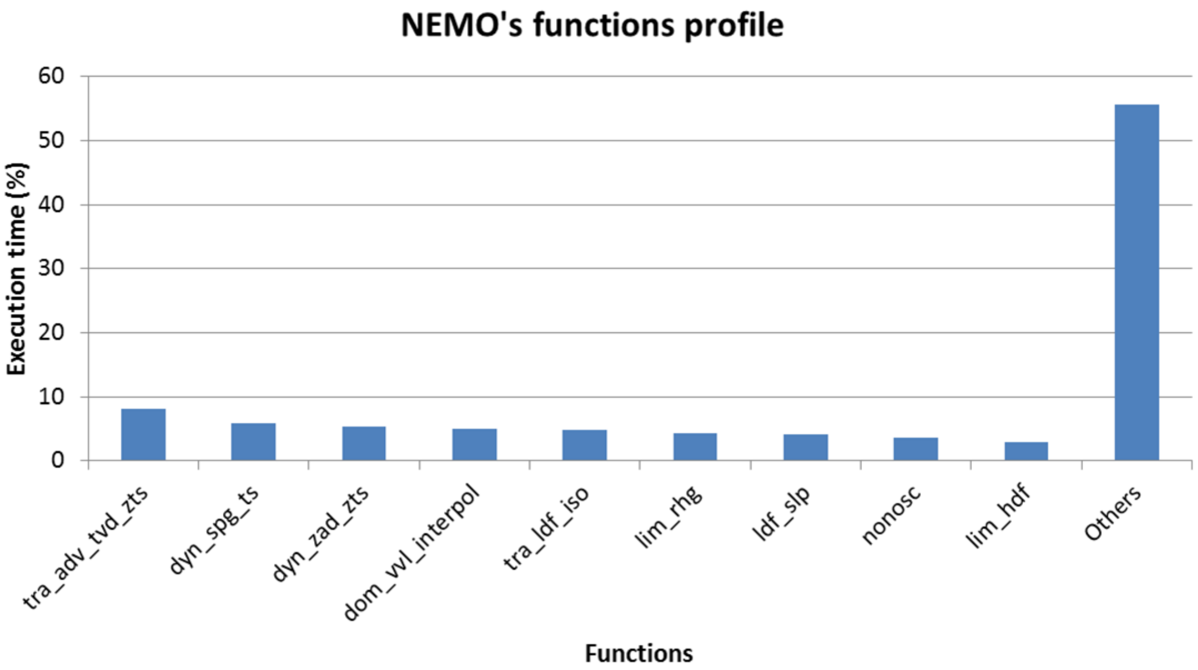


Figure 17: Profile of the execution time of the largest computational cost of NEMO's functions

The case of NEMO is very similar to IFS: the execution time is very distributed between many functions, so according to Amdahl's law[2], probably it is not worth the effort to optimize a function included in the *Others* group. Nevertheless, there are some functions included in the chart such as *tra_adv_tvd_zts*, *dyn_spg_ts* or *lim_rhg* that should be taken into account.

[2] Applying Amdahl's law, it is not worth to optimize (even infinitely faster, i.e., executed instantly) a region of code that represents a very small percentage of the total execution time, because the reduction in the total execution time will be negligible.

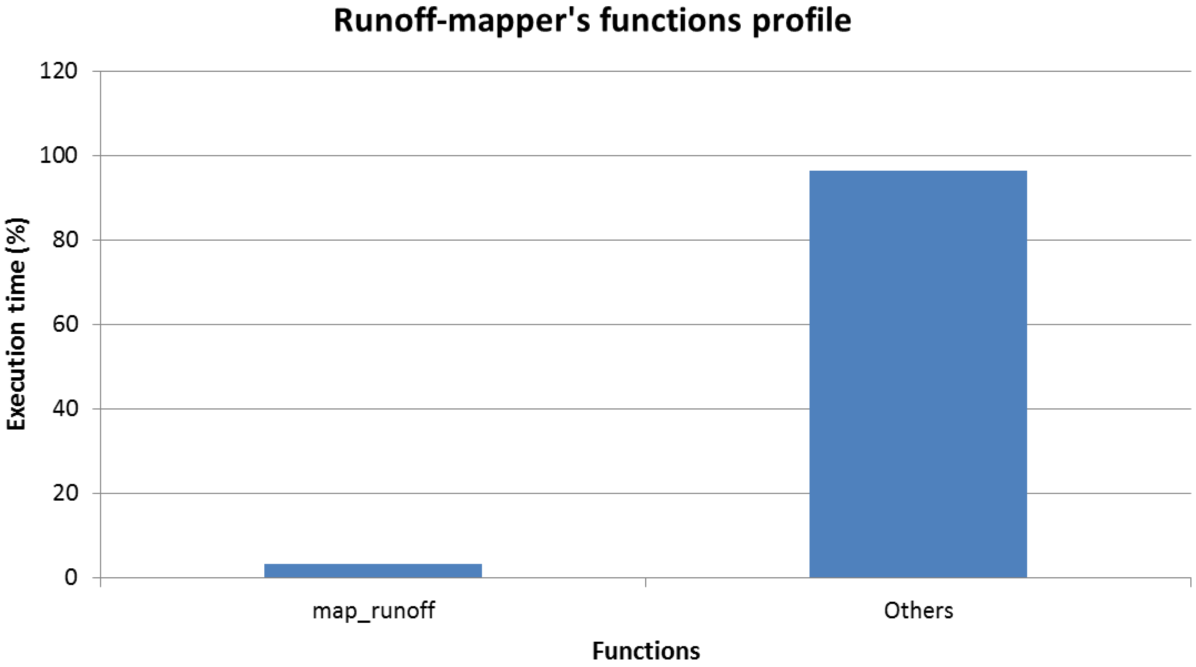


Figure 18: Profile of the execution time of the largest computational cost of runoff-mapper's functions

Finally, the *Others* column of runoff-mapper chart corresponds mainly to OASIS3-MCT's functions, so it is evident that the coupler is a key component for the performance of EC-Earth.

4.3. Tracing

In this section we will go through two subsections: in the first one we show the structure of EC-Earth using 64 MPI processes for IFS and 16 for NEMO; in the second subsection we explain which and where are the bottlenecks of EC-Earth using the fastest combination, i.e., 512 MPI processes for IFS and 128 for NEMO.

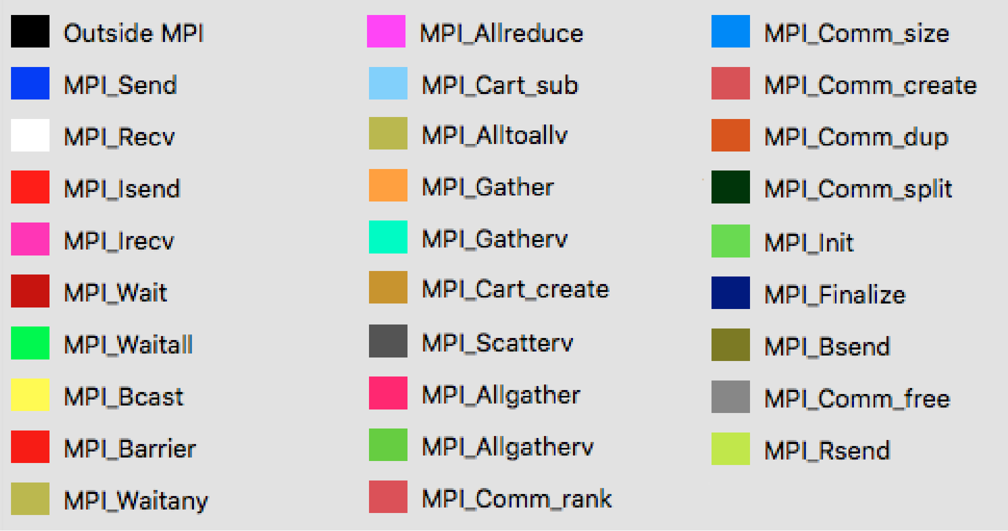
4.3.1. Basic analysis of the structure

It is important to understand how the execution of EC-Earth is organized to focus on the important issues later. A first approach of the structure can be analyzed with a small combination of MPI processes, such as 64 for IFS and 16 for NEMO. This is enough to build an overview of EC-Earth.

A trace has the following structure: on the y axis there are the different MPI processes of EC-Earth, starting with the IFS ones, NEMO, runoff-mapper and the last one is XIOS; the x axis represents the timeline. Along this timeline, some (or many) events happen, which can be related to MPI calls, cache misses, MIPS and many other performance metrics. These events can be chosen with Paraver, and depends on the configuration of Extrae, i.e., we have to write in an XML file the events that we want to generate.

Figure 21 shows a complete trace of EC-Earth performing a one day simulation. There are three main parts in the structure: initialization, simulation and finalization. We will focus only on the simulation part, because the initialization and finalization parts become insignificant in real simulations. For example, for a chunk of one simulated year, they represent about 1.6% of the final execution time.

In Figure 21 there are two traces, which have exactly the same MPI processes and timeline. The trace (a) shows the different MPI calls of the processes and the trace (b) the useful duration. For traces of MPI calls, each color means an MPI state or outside MPI (useful execution). Figure 19 shows the legend of the different MPI calls used by Paraver.



■ Outside MPI	■ MPI_Allreduce	■ MPI_Comm_size
■ MPI_Send	■ MPI_Cart_sub	■ MPI_Comm_create
□ MPI_Recv	■ MPI_Alltoallv	■ MPI_Comm_dup
■ MPI_Isend	■ MPI_Gather	■ MPI_Comm_split
■ MPI_Irecv	■ MPI_Gatherv	■ MPI_Init
■ MPI_Wait	■ MPI_Cart_create	■ MPI_Finalize
■ MPI_Waitall	■ MPI_Scatterv	■ MPI_Bsend
■ MPI_Bcast	■ MPI_Allgather	■ MPI_Comm_free
■ MPI_Barrier	■ MPI_Allgatherv	■ MPI_Rsend
■ MPI_Waitany	■ MPI_Comm_rank	

Figure 19: Legend with the colors representation of the MPI calls used by Paraver

The simulation part of the Figure 21 has a repetitive pattern which corresponds to the time steps of the model. The useful duration trace shows the size of the useful computation chunks, the more time computing without interruptions, the more useful computation is considered. Figure 20 shows the color scale for useful duration traces. Left colors denote very small computation chunks, whereas right colors denote large chunks. So the scale is in ascending order. In case of nothing useful, it is painted in black.

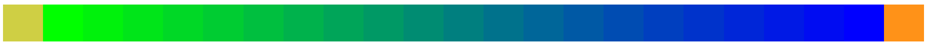


Figure 20: Color scale for useful duration traces used by Paraver

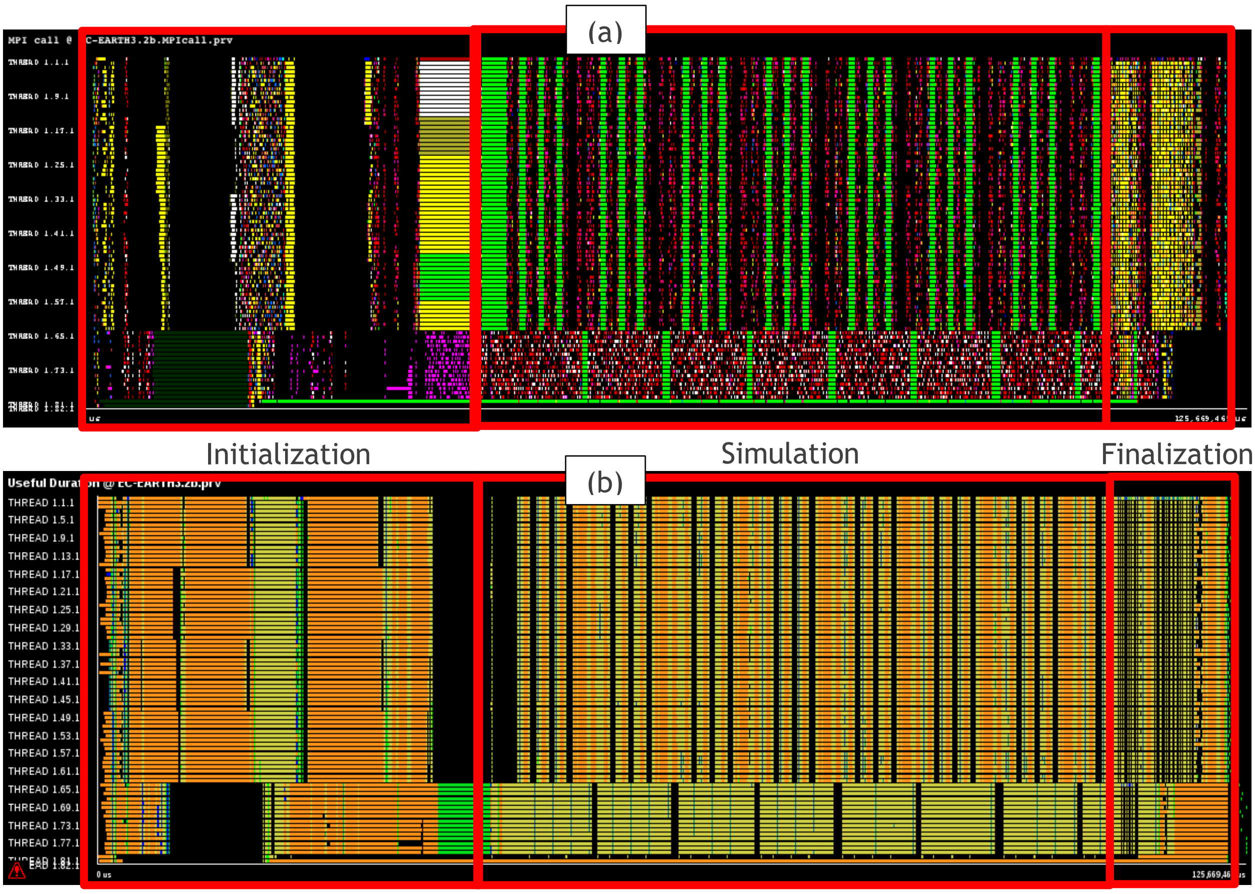

(a)
MPI call @ EC-EARTH3.2b.MPIcall.prv
Initialization
Simulation
Finalization
(b)
Useful Duration @ EC-EARTH3.2b.prv
0 us
125,669,46 us

Figure 21: Complete execution of one day of simulation using 64 MPI processes for IFS and 16 for NEMO. Trace (a) shows the MPI calls and trace (b) shows the useful duration.

In spite of the repetitive pattern in the simulation part, note that these patterns are not completely equal, as it can be appreciated in Figure 22. While in NEMO all time steps are equal, in IFS one of every four time steps is different (according to the configuration used, this frequency could change), because it is needed to execute radiation routines. For this particular combination of MPI processes, taking into account that synchronization is needed at the end of each time step between coupled components, IFS has to wait for NEMO in regular time steps, except when it has to execute radiation routines, in this case NEMO has to wait for IFS to finish.

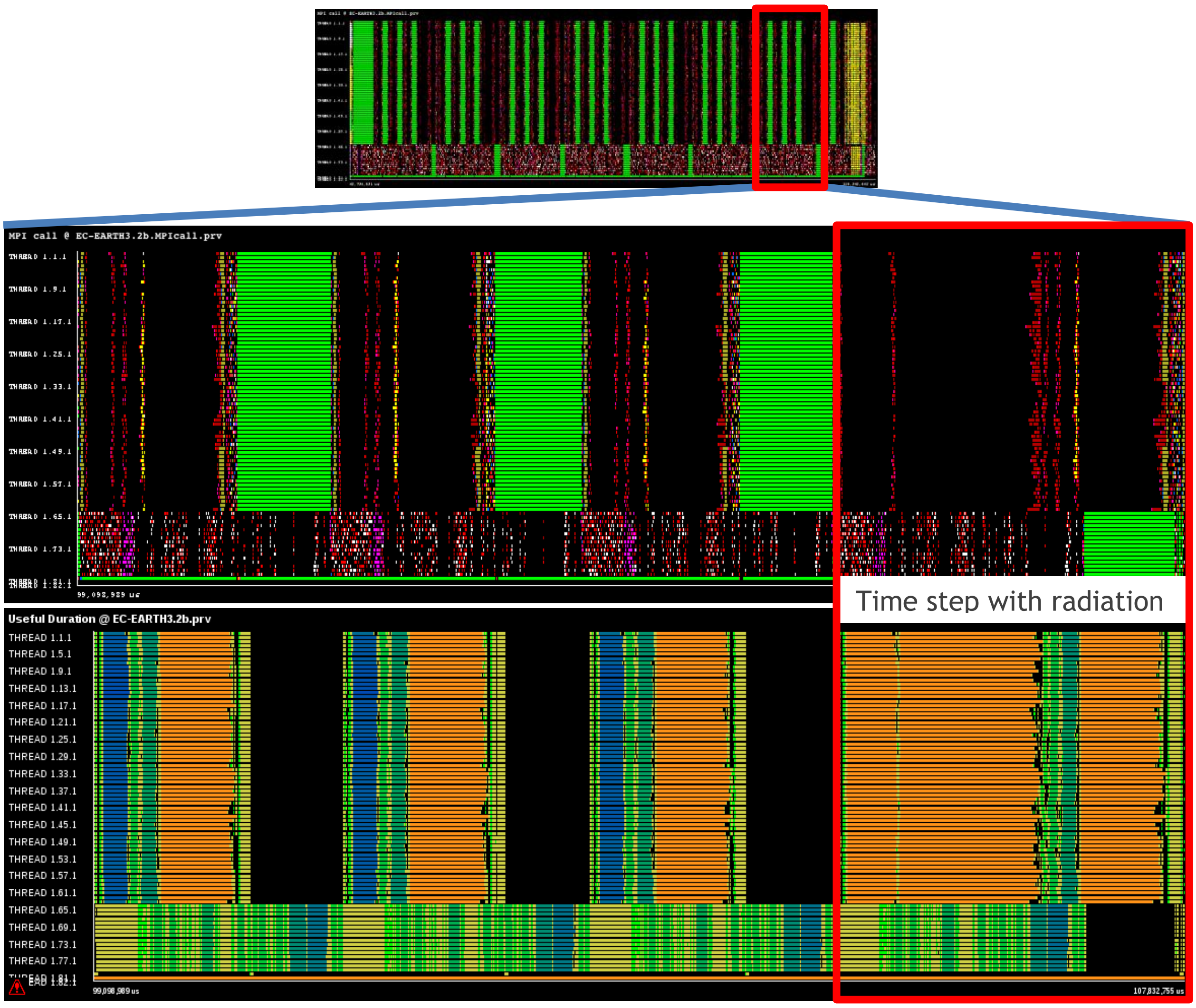


Figure 22: Traces with four time steps. All NEMO time steps are equal, but in the IFS ones, one every four takes more time as it executes radiation routines.

Finally, the tracing study reveals one bottleneck of the coupled model: at the end of each IFS time step, IFS needs to execute some OASIS3-MCT's functions to send some variables to NEMO. In this process there are a lot of inefficient communications without useful computation. Figure 23 shows this problem, where can be appreciated many communications, mainly MPI_Recv and MPI_Bcast. Nevertheless, this particular small combination of MPI processes does not represent a huge problem. The useful duration trace shows perfectly that almost nothing useful is computed, basically because they are waiting to communicate data.

In section 4.3.2 we deal with this issue in detail.

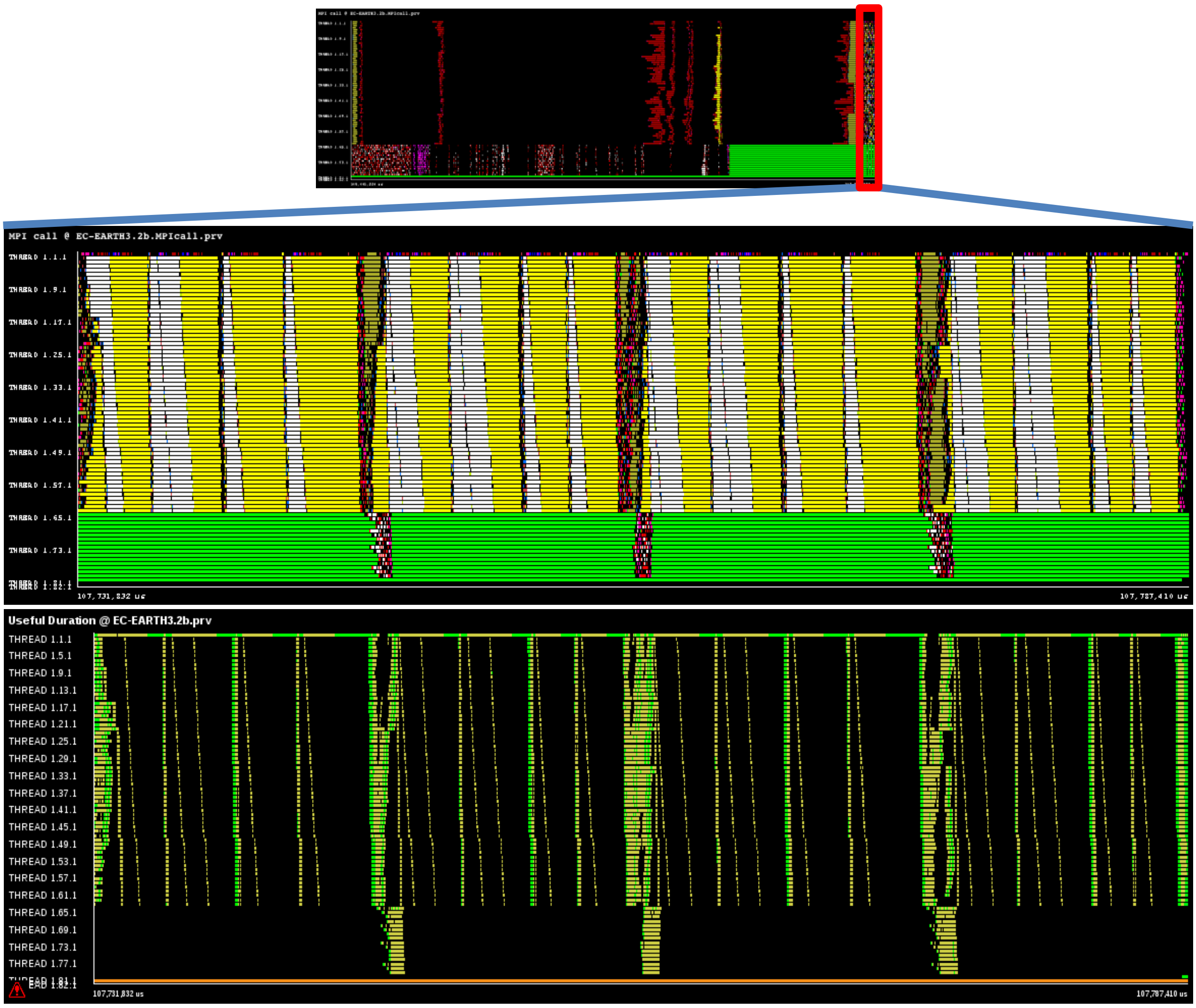


Figure 23: The traces show that at the end of each IFS time step there are many communications with almost no useful computation.

4.3.2. Detailed analysis

For the detailed analysis we use the fastest combination, having 512 MPI processes for IFS and 128 for NEMO. This is a combination where the parallel overhead increases and some important issues are appreciated, such as a potential serialization, briefly commented in the previous section.

To introduce how EC-Earth works, it is interesting to see the correlation between IFS and NEMO time steps. Figure 24 shows in different colors the identifiers, i.e., the number of time step. For example, dark blue represents the first time step, white represents the second time step, and so on. It is deducible that NEMO goes one time step ahead.

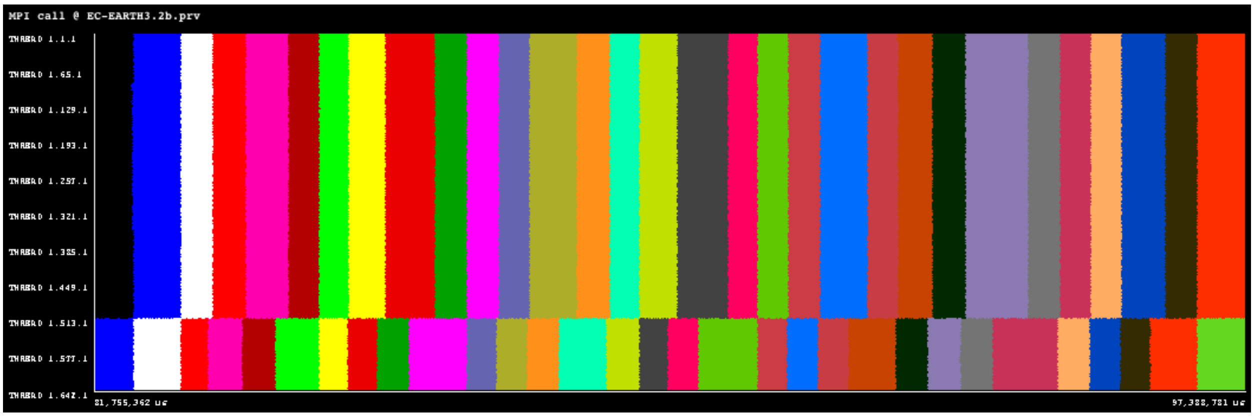


Figure 24: Colors used to identify the IDs of both IFS and NEMO time steps.

We previously mentioned that one of every four time steps of IFS executes radiation routines, so this only represents the 25% of the time steps of IFS. For this reason and because radiation routines seem to not have relevant bottlenecks, we will reduce our analysis to time steps without radiation.

Figure 25 shows a complete time step of EC-Earth, including the 512 MPI processes of IFS, 128 of NEMO, one of runoff-mapper and finally one of XIOS. Trace (b) of Figure 25 shows that the second half of IFS is completely useless in terms of useful computation, as the processes are sending the required data to NEMO through OASIS3-MCT. This region of the trace corresponds to OASIS code, which is doing the coupling from IFS to NEMO. This coupling includes an interpolation from the source grid to the target grid and applies to some fields an expensive conservative process of transformation.

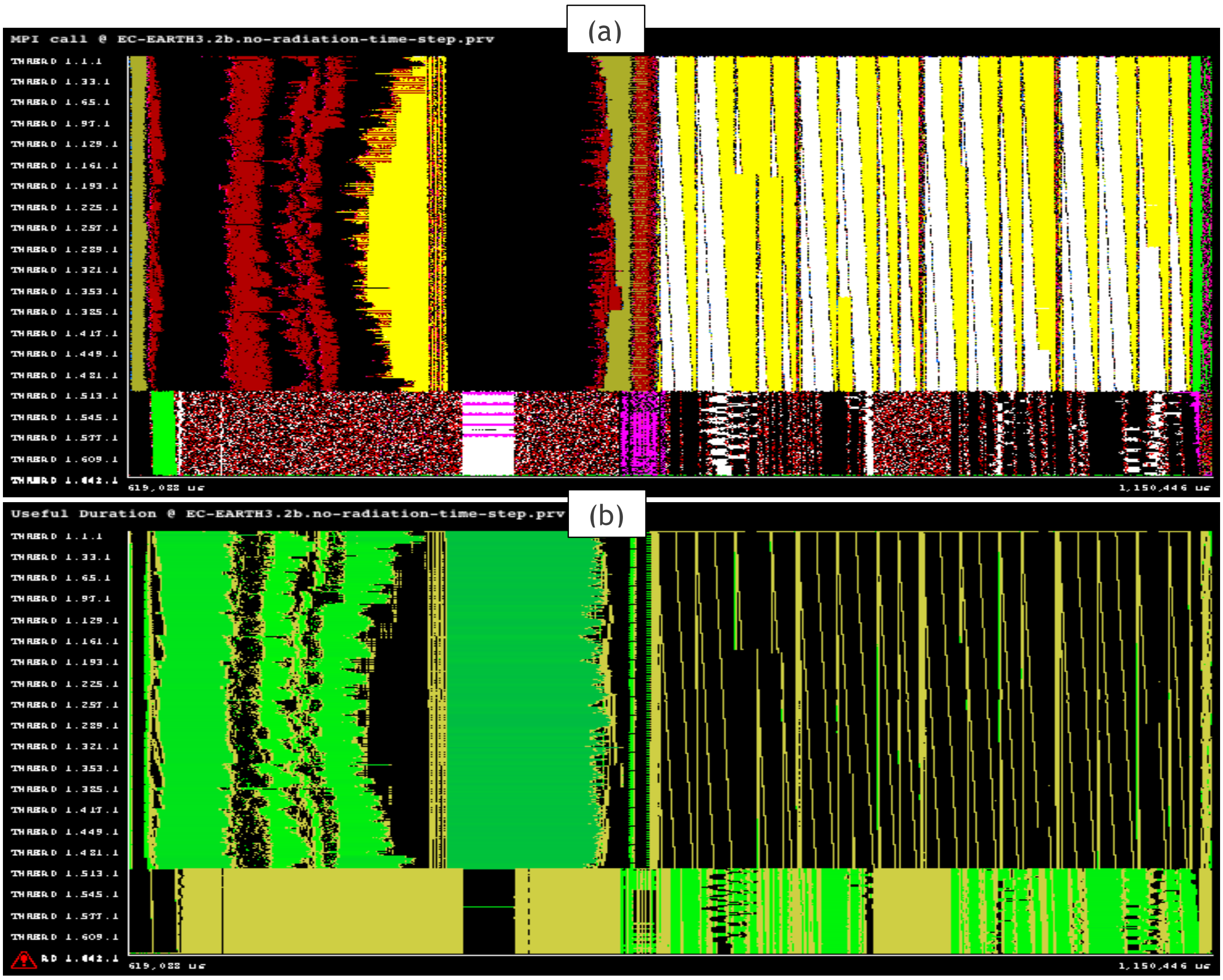


Figure 25: General overview of one time step of EC-Earth, including one time step of IFS without radiation.

After applying the coupling method to several variables (some of them in a conservative process), IFS can send the data to NEMO, which is represented with the red lines of trace (a) of Figure 26. By contrast, trace (b) shows the data sent from NEMO to IFS also through OASIS3-MCT. In this case, the coupling method is not conservative in any case, so the efficiency impact is almost negligible.

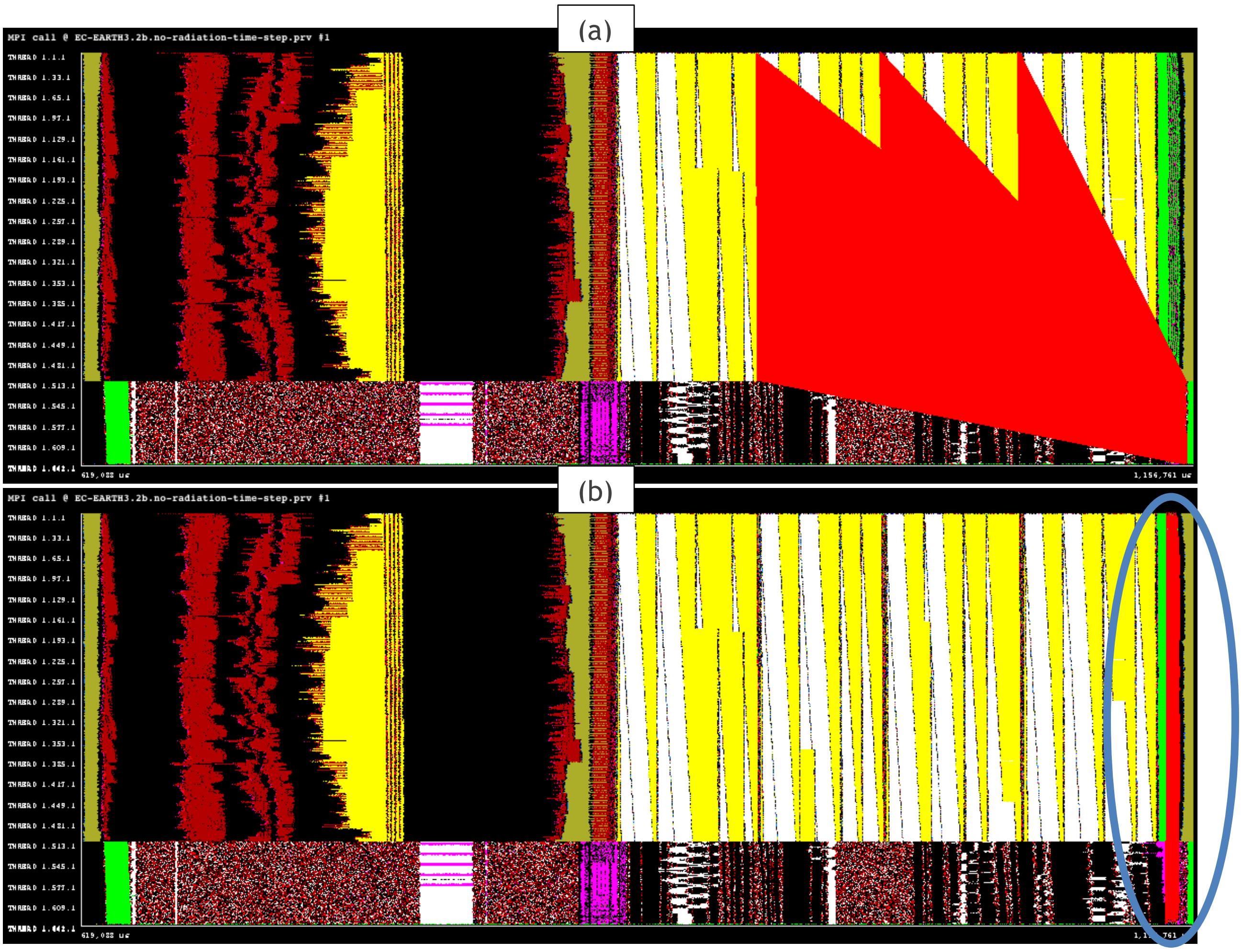


Figure 26: Communications between IFS and NEMO through OASIS3-MCT. Trace (a) shows the data sent from IFS to NEMO. In contrast, trace (b) shows the data sent from NEMO to IFS.

4.3.2.1. IFS issues

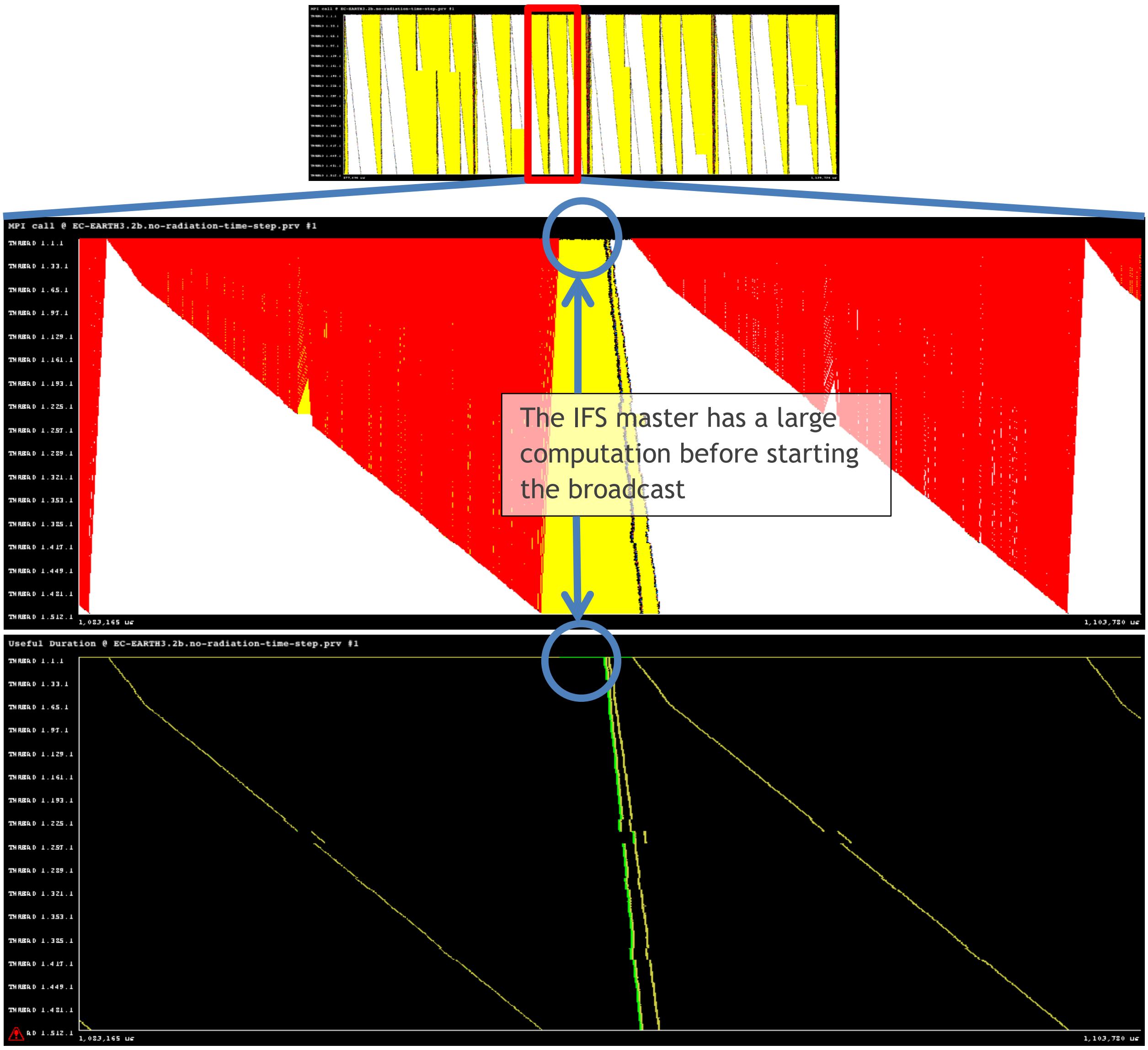
4.3.2.1.1. Serialization due to the conservative method

As we previously mentioned, the conservative method used by OASIS3-MCT has important consequences, because the way it works is comparable to execute the code sequentially. This is the main issue to face in IFS. In Figure 27 can be appreciated in detail how the serialization is applied: the IFS master process sends a point-to-point message to the rest of IFS processes and then it receives a message from each one of them. This pattern is repeated 24 times in each time step. Furthermore, after 16 of these point-to-point patterns, all the processes execute 3 broadcasts in a row, but before of them, the IFS master has a large computation where all the processes have to wait.

In summary, the serialization is due to the IFS master process because it is responsible for performing the complete coupling (using OASIS3-MCT). For this process, the IFS master follows these steps:

1) Receives the data fields from the rest of IFS processes.
2) Does the transformation process from the source grid to the target grid.
3) Makes some corrections for conservative fields.
4) Sends transformed target grid data to each IFS process.

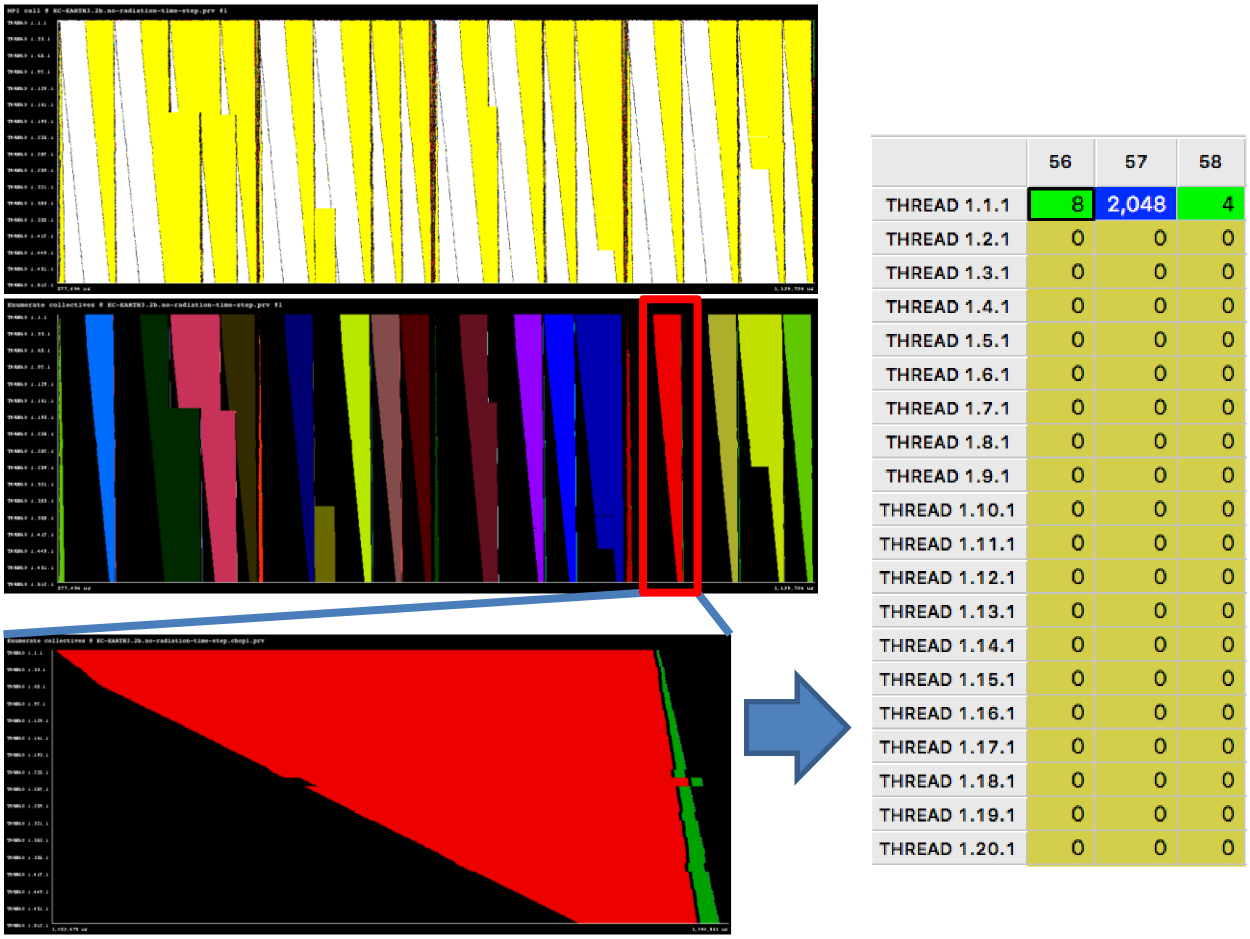
Finally, each IFS process sends its subdomain to the target component, NEMO in this case.



The IFS master has a large computation before starting the broadcast

Figure 27: Detail of the point-to-point communications of the IFS master with the rest of IFS processes. In addition, there are some broadcasts and before of them, the IFS master has a large computation where everyone is blocked.

Figure 28 has a trace that is used to color the different collective MPI calls from the whole conservative part. Focusing on just 3 broadcasts, we enumerate them and show that the sizes are small: 8, 2048 and 4 bytes. Therefore, broadcasts not seem to have a significant impact on the performance of IFS.



	56	57	58
THREAD 1.1.1	8	2,048	4
THREAD 1.2.1	0	0	0
THREAD 1.3.1	0	0	0
THREAD 1.4.1	0	0	0
THREAD 1.5.1	0	0	0
THREAD 1.6.1	0	0	0
THREAD 1.7.1	0	0	0
THREAD 1.8.1	0	0	0
THREAD 1.9.1	0	0	0
THREAD 1.10.1	0	0	0
THREAD 1.11.1	0	0	0
THREAD 1.12.1	0	0	0
THREAD 1.13.1	0	0	0
THREAD 1.14.1	0	0	0
THREAD 1.15.1	0	0	0
THREAD 1.16.1	0	0	0
THREAD 1.17.1	0	0	0
THREAD 1.18.1	0	0	0
THREAD 1.19.1	0	0	0
THREAD 1.20.1	0	0	0

Figure 28: The last two traces are used to enumerate the collective MPI calls, basically broadcasts. These small broadcasts are from the conservative part of OASIS3-MCT. There are groups of three which have sizes of 8, 2048 and 4 bytes.

4.3.2.1.2. Workload imbalance

The other problem of IFS is a load imbalance due to the Semi-Lagrangian treatment of advection in physical space. For a distributed memory parallelization, this calculation requires access to the grid-point data held on neighboring processors, i.e., each subdomain has available a halo space where this data is stored, so message passing is required to obtain it.

The Semi-Lagrangian treatment is done in two steps:

- In the first part, for each model time step the full halo for fields needed for calculating the departure point is constructed and communicated.
- The second part uses an “on-demand” scheme to communicate only the required halo points for fields used in the interpolations. The message passing in the second part is done in 2 steps as well:
    - First, a list of points required by a given processor is communicated to the surrounding processors.
    - After that, the required points are communicated back from the surrounding processors to the requesting processor.

The computational behavior of this implementation can be observed in Figure 29 where a load imbalance among external and internal processes occurs, because internal MPI processes have less work to execute. This can be checked by the histogram of correlated instructions and instructions per cycle (IPC). Ideally, points should be vertically aligned, but they are not.

This load imbalance could be related to the domain decomposition done for physical space, where the grid is divided in latitude and longitude ranges. Using this decomposition, subdomains on the poles could be significantly different to subdomains close to the equator. Actually, Figure 29 shows that external processes (close to the poles) have a similar behavior. On the other hand, internal processes (close to the equator) also have a similar behavior. It could be that external processes close to the poles had more neighbors due to the complex domain decomposition in that zone, increasing the computational load and MPI communications of the Semi-Lagrangian method for these processes.

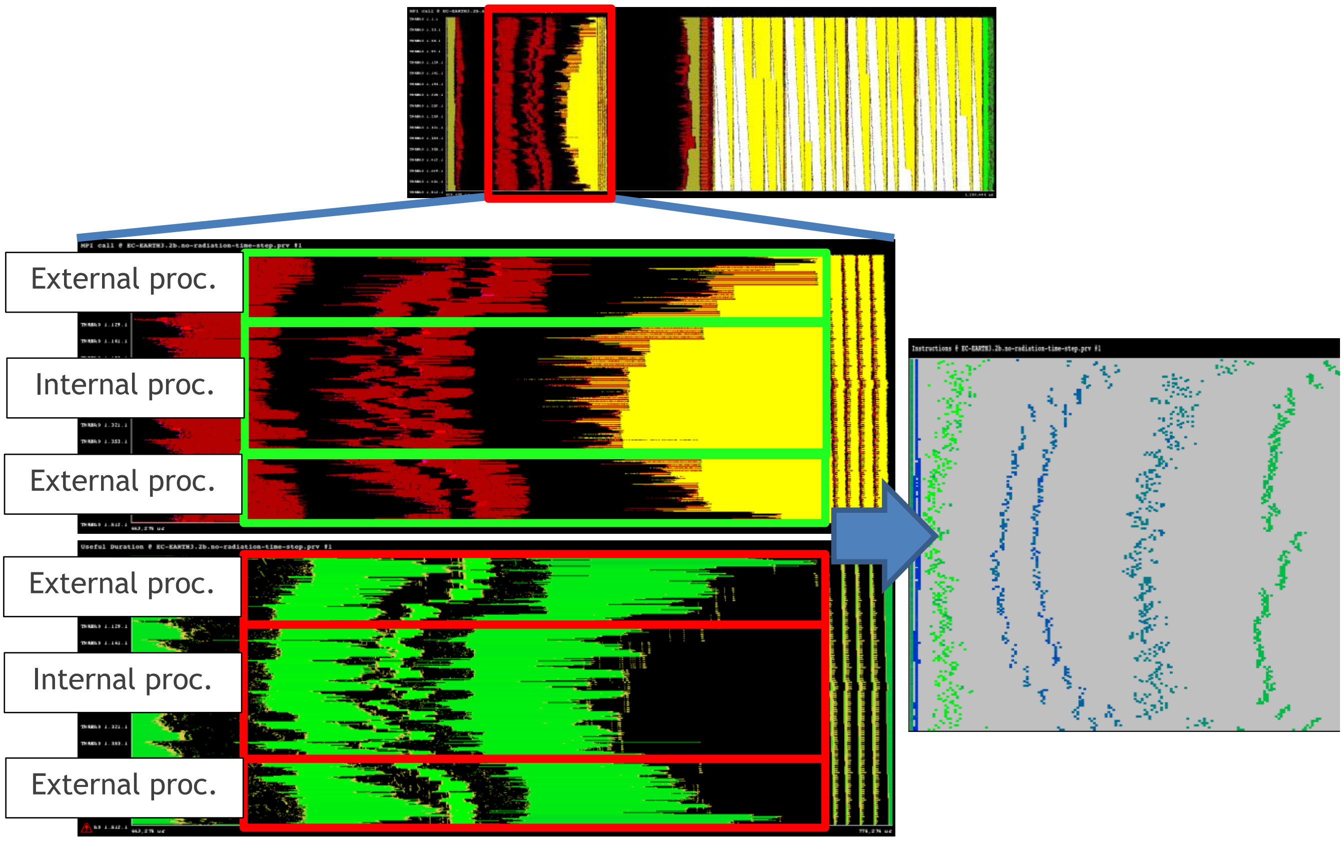
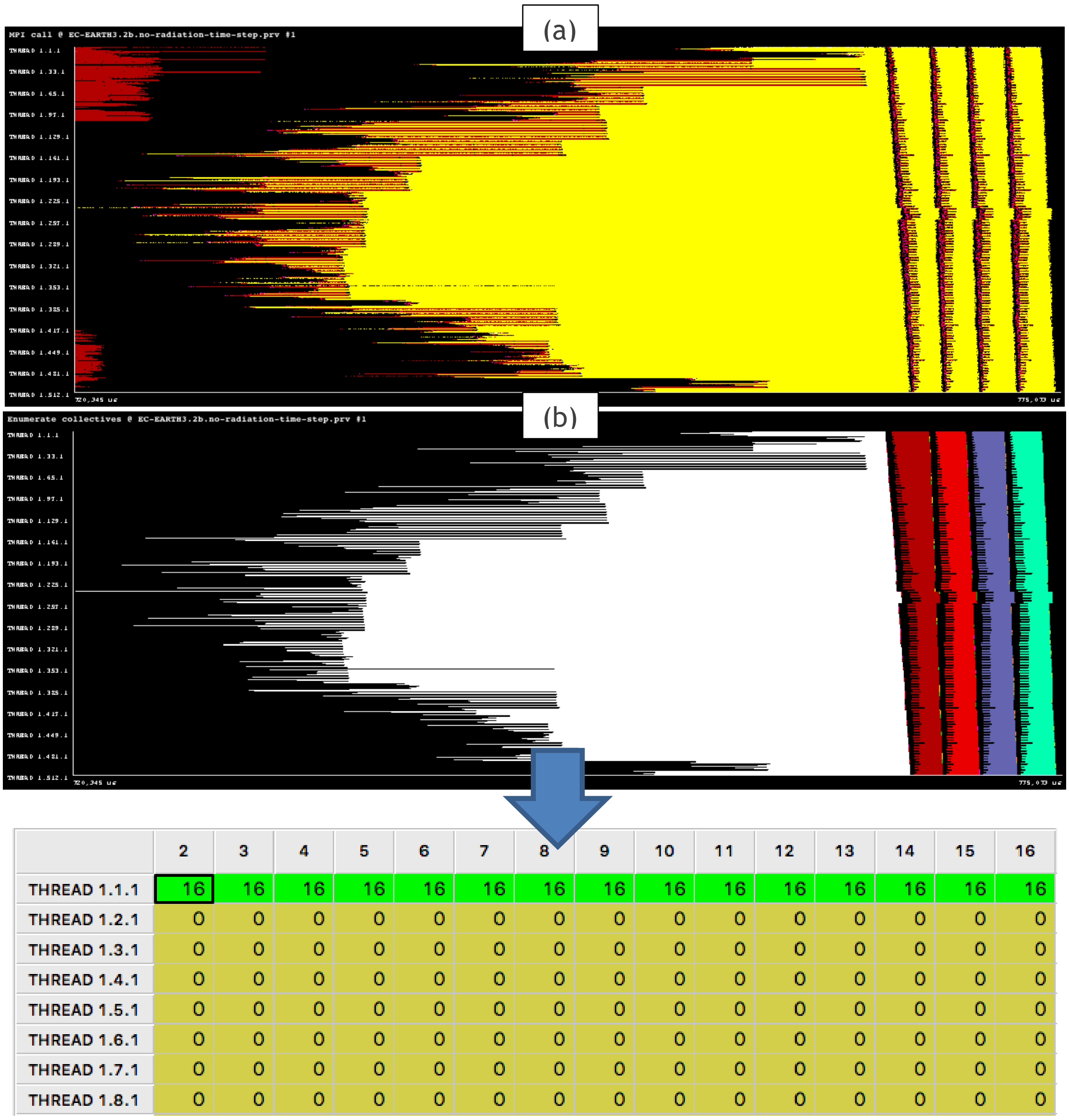


Figure 29: IFS time step has an imbalance between the beginning and the middle. The internal processes have less work as it can be seen in the histogram of correlated instructions and IPC (on the right image). External MPI processes have more instructions to execute.

After the Semi-Lagrangian method, there are some broadcasts that might seem to contribute to increase the parallel overhead of the model with collective communications. However, these communications are very small, only 16 bytes each one. They do not have a significant impact on the performance of the model. Figure 30 shows the broadcasts, using an MPI call trace (a) and a collective MPI call trace (b). The table indicates the amount of bytes sent by each process in each collective MPI call. This means that the IFS master process is sending 16 bytes to each one of the rest of IFS processes in each broadcast.

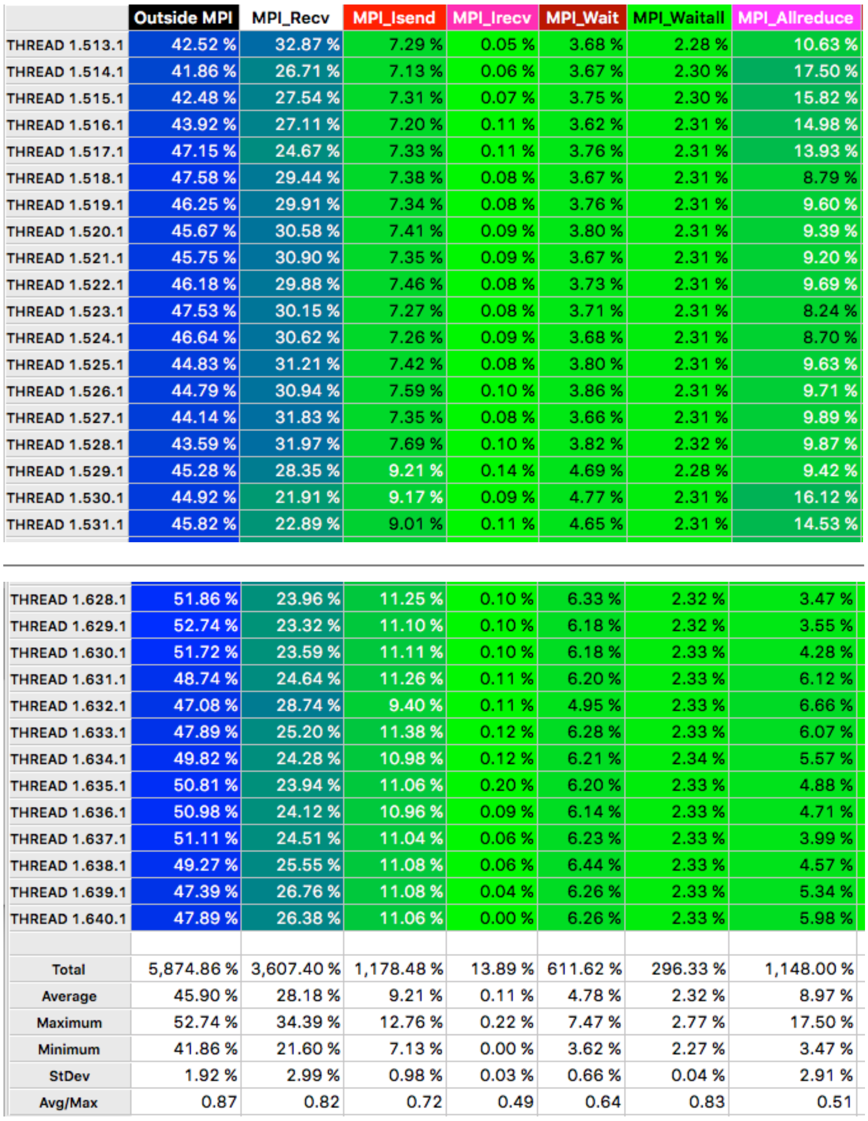


	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
THREAD 1.1.1	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
THREAD 1.2.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THREAD 1.3.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THREAD 1.4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THREAD 1.5.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THREAD 1.6.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THREAD 1.7.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
THREAD 1.8.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30: After the imbalance of IFS there are some broadcasts that could seem to have an important impact, but actually, they do not as their size is only 16 bytes each one.

4.3.2.2. NEMO issues

The main problem in the execution of NEMO is its communication overhead. The tendency of NEMO processes is to compute very small bursts and communicate with neighboring processes. This has a huge overhead, so at the end the useful computation is reduced drastically. Figure 31 shows a partial MPI call profile of some MPI processes of NEMO and their summary (the execution is using 128 processes). Note that the first process starts with the ID 513, because the first 512 are IFS processes. The first column of the table, called *Outside MPI*, shows the time spent executing something useful, i.e., the MPI library is not used. The rest of columns show the most executed MPI functions. Thus, we can deduce that NEMO is only computing about 45.9% of the execution time in each time step (Average of first column, Outside MPI). This means that communications represent an important part of the overhead introduced for the parallel execution of NEMO.



First NEMO processes

	Outside MPI	MPI_Recv	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce
THREAD 1.513.1	42.52 %	32.87 %	7.29 %	0.05 %	3.68 %	2.28 %	10.63 %
THREAD 1.514.1	41.86 %	26.71 %	7.13 %	0.06 %	3.67 %	2.30 %	17.50 %
THREAD 1.515.1	42.48 %	27.54 %	7.31 %	0.07 %	3.75 %	2.30 %	15.82 %
THREAD 1.516.1	43.92 %	27.11 %	7.20 %	0.11 %	3.62 %	2.31 %	14.98 %
THREAD 1.517.1	47.15 %	24.67 %	7.33 %	0.11 %	3.76 %	2.31 %	13.93 %
THREAD 1.518.1	47.58 %	29.44 %	7.38 %	0.08 %	3.67 %	2.31 %	8.79 %
THREAD 1.519.1	46.25 %	29.91 %	7.34 %	0.08 %	3.76 %	2.31 %	9.60 %
THREAD 1.520.1	45.67 %	30.58 %	7.41 %	0.09 %	3.80 %	2.31 %	9.39 %
THREAD 1.521.1	45.75 %	30.90 %	7.35 %	0.09 %	3.67 %	2.31 %	9.20 %
THREAD 1.522.1	46.18 %	29.88 %	7.46 %	0.08 %	3.73 %	2.31 %	9.69 %
THREAD 1.523.1	47.53 %	30.15 %	7.27 %	0.08 %	3.71 %	2.31 %	8.24 %
THREAD 1.524.1	46.64 %	30.62 %	7.26 %	0.09 %	3.68 %	2.31 %	8.70 %
THREAD 1.525.1	44.83 %	31.21 %	7.42 %	0.08 %	3.80 %	2.31 %	9.63 %
THREAD 1.526.1	44.79 %	30.94 %	7.59 %	0.10 %	3.86 %	2.31 %	9.71 %
THREAD 1.527.1	44.14 %	31.83 %	7.35 %	0.08 %	3.66 %	2.31 %	9.89 %
THREAD 1.528.1	43.59 %	31.97 %	7.69 %	0.10 %	3.82 %	2.32 %	9.87 %
THREAD 1.529.1	45.28 %	28.35 %	9.21 %	0.14 %	4.69 %	2.28 %	9.42 %
THREAD 1.530.1	44.92 %	21.91 %	9.17 %	0.09 %	4.77 %	2.31 %	16.12 %
THREAD 1.531.1	45.82 %	22.89 %	9.01 %	0.11 %	4.65 %	2.31 %	14.53 %

Last NEMO processes

	Outside MPI	MPI_Recv	MPI_Isend	MPI_Irecv	MPI_Wait	MPI_Waitall	MPI_Allreduce
THREAD 1.628.1	51.86 %	23.96 %	11.25 %	0.10 %	6.33 %	2.32 %	3.47 %
THREAD 1.629.1	52.74 %	23.32 %	11.10 %	0.10 %	6.18 %	2.32 %	3.55 %
THREAD 1.630.1	51.72 %	23.59 %	11.11 %	0.10 %	6.18 %	2.33 %	4.28 %
THREAD 1.631.1	48.74 %	24.64 %	11.26 %	0.11 %	6.20 %	2.33 %	6.12 %
THREAD 1.632.1	47.08 %	28.74 %	9.40 %	0.11 %	4.95 %	2.33 %	6.66 %
THREAD 1.633.1	47.89 %	25.20 %	11.38 %	0.12 %	6.28 %	2.33 %	6.07 %
THREAD 1.634.1	49.82 %	24.28 %	10.98 %	0.12 %	6.21 %	2.34 %	5.57 %
THREAD 1.635.1	50.81 %	23.94 %	11.06 %	0.20 %	6.20 %	2.33 %	4.88 %
THREAD 1.636.1	50.98 %	24.12 %	10.96 %	0.09 %	6.14 %	2.33 %	4.71 %
THREAD 1.637.1	51.11 %	24.51 %	11.04 %	0.06 %	6.23 %	2.33 %	3.99 %
THREAD 1.638.1	49.27 %	25.55 %	11.08 %	0.06 %	6.44 %	2.33 %	4.57 %
THREAD 1.639.1	47.39 %	26.76 %	11.08 %	0.04 %	6.26 %	2.33 %	5.34 %
THREAD 1.640.1	47.89 %	26.38 %	11.06 %	0.00 %	6.26 %	2.33 %	5.98 %
Total	5,874.86 %	3,607.40 %	1,178.48 %	13.89 %	611.62 %	296.33 %	1,148.00 %
Average	45.90 %	28.18 %	9.21 %	0.11 %	4.78 %	2.32 %	8.97 %
Maximum	52.74 %	34.39 %	12.76 %	0.22 %	7.47 %	2.77 %	17.50 %
Minimum	41.86 %	21.60 %	7.13 %	0.00 %	3.62 %	2.27 %	3.47 %
StDev	1.92 %	2.99 %	0.98 %	0.03 %	0.66 %	0.04 %	2.91 %
Avg/Max	0.87	0.82	0.72	0.49	0.64	0.83	0.51

Figure 31: MPI call profile of some MPI processes of NEMO and the whole summary. In average, the processes only compute a 45.9% of the execution time in each time step.

Furthermore, NEMO also has some other smaller problems than the one explained above, which are needed to be taken into account. Figure 32 highlights these problems, which are also listed below:

- An initial waiting due to runoff-mapper component.
- Preemption[3].
- A large global communication.
- Some important imbalances.

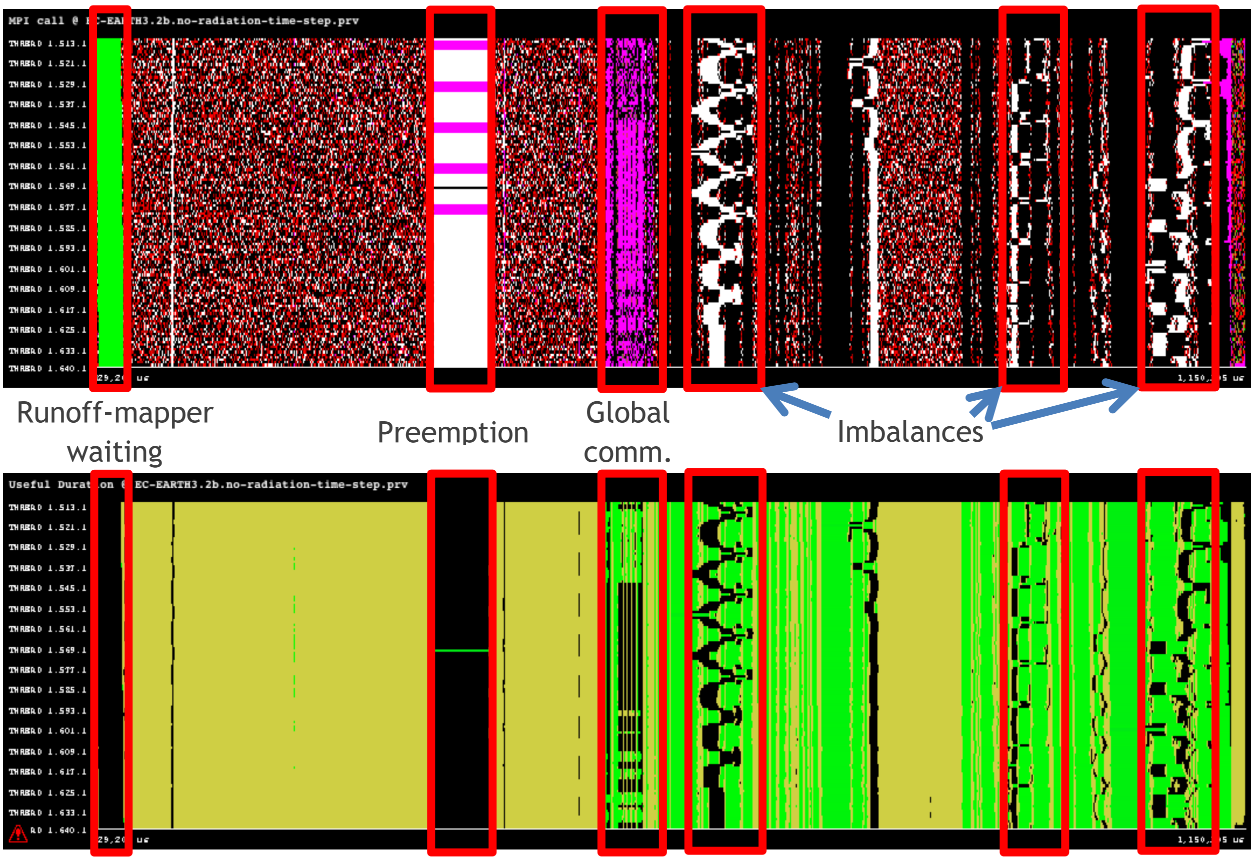

MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
Runoff-mapper waiting
Preemption
Global comm.
Imbalances
Useful Duration @ EC-EARTH3.2b.no-radiation-time-step.prv

Figure 32: Global view of one NEMO time step. There are some issues: waiting, preemption, global communications and imbalances.

[3] **Preemption** is the ability of the operating system of interrupting a task being carried out in favor of a higher priority task. These changes of tasks are known as context switches.

4.3.2.2.1. Waiting for runoff-mapper

To understand the first issue, it is interesting to show graphically how runoff-mapper communicates with IFS and NEMO. Trace (a) of Figure 33 shows the communications going from IFS to runoff-mapper, while trace (b) shows the communication from runoff-mapper to NEMO.

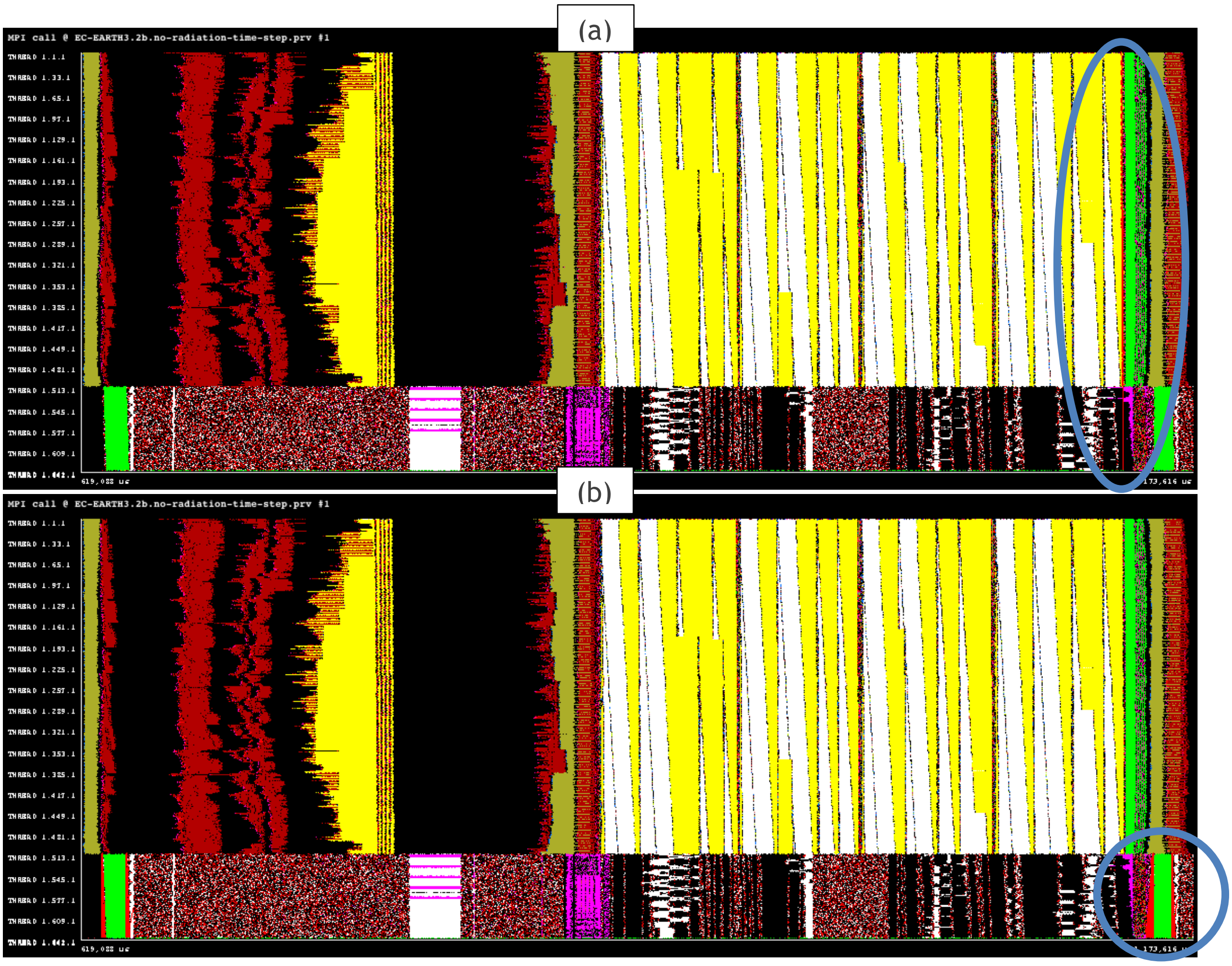


Figure 33: Communications from IFS to NEMO through runoff-mapper. Trace (a) shows the data sent from IFS to runoff-mapper and trace (b) shows the data sent from runoff-mapper to NEMO.

The problem appears in the communication from runoff-mapper to NEMO, as NEMO has to wait for runoff-mapper to finish a large computation before sending the data.

Figure 34 has four traces: the first one (a) is the whole time step of EC-Earth, whereas the second one (b) is a zoom of the first one focusing on the issue. The last two traces are only showing the runoff-mapper's timeline of MPI call (c) and useful duration (d) respectively. Their time scale exactly corresponds to the trace (b).

Runoff-mapper sends data to NEMO in two batches, corresponding to two sections of yellow lines of trace (b), and also indicated in the trace (c). Before each batch there is a large computation, which are indicated in trace (d), so NEMO has to wait between the first and the second batch of sends using one MPI_Waitall for each process, which is indicated in trace (b) with green color.

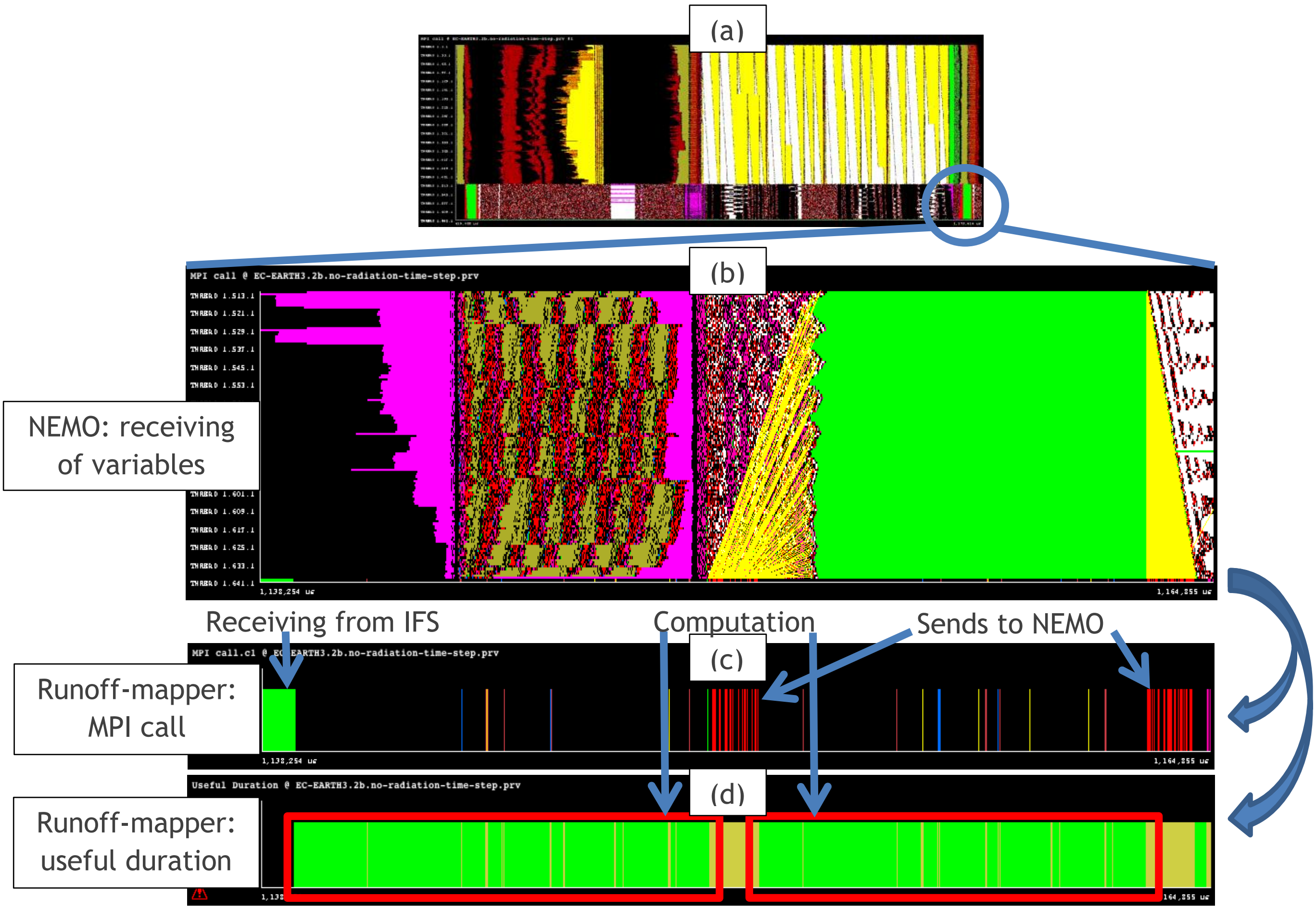
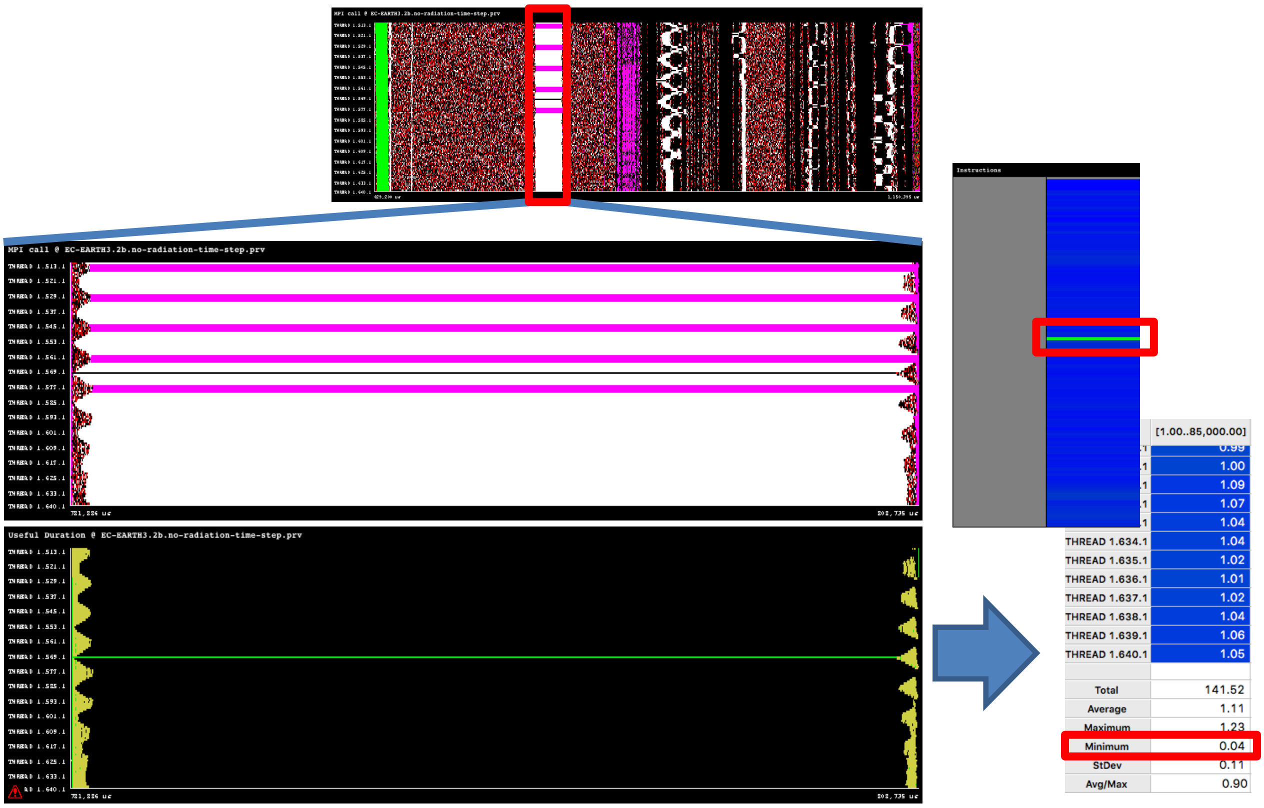

(a)
(b)
MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
NEMO: receiving of variables
1,132,254 us
1,164,255 us
Receiving from IFS
Computation
Sends to NEMO
(c)
MPI call.c1 @ EC-EARTH3.2b.no-radiation-time-step.prv
Runoff-mapper: MPI call
(d)
Useful Duration @ EC-EARTH3.2b.no-radiation-time-step.prv
Runoff-mapper: useful duration

Figure 34: Detail of data communication from runoff-mapper to NEMO. After receiving the data from IFS, runoff-mapper sends the variables to NEMO in two batches, but before each one there are large computation chunks. The second chunk is causing the waiting in NEMO processes.

4.3.2.2.2. Preemption

The second problem is preemption, caused by the operating system because it needs to execute another process. Despite being possible to fix this[4], it is not part of this study. Figure 35 shows this problem, where one MPI process has a large computation chunk with a very low IPC, because at the same time, the core is executing another process (or processes) without being related to EC-Earth.

Nevertheless, this does not happen frequently, so it is not an important problem. For example, if we trace a NEMO time step again, probably we would not get this preemption.


MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
Useful Duration @ EC-EARTH3.2b.no-radiation-time-step.prv
Instructions

	[1.00..85,000.00]
.1	0.99
.1	1.00
.1	1.09
.1	1.07
.1	1.04
THREAD 1.634.1	1.04
THREAD 1.635.1	1.02
THREAD 1.636.1	1.01
THREAD 1.637.1	1.02
THREAD 1.638.1	1.04
THREAD 1.639.1	1.06
THREAD 1.640.1	1.05
Total	141.52
Average	1.11
Maximum	1.23
Minimum	0.04
StDev	0.11
Avg/Max	0.90

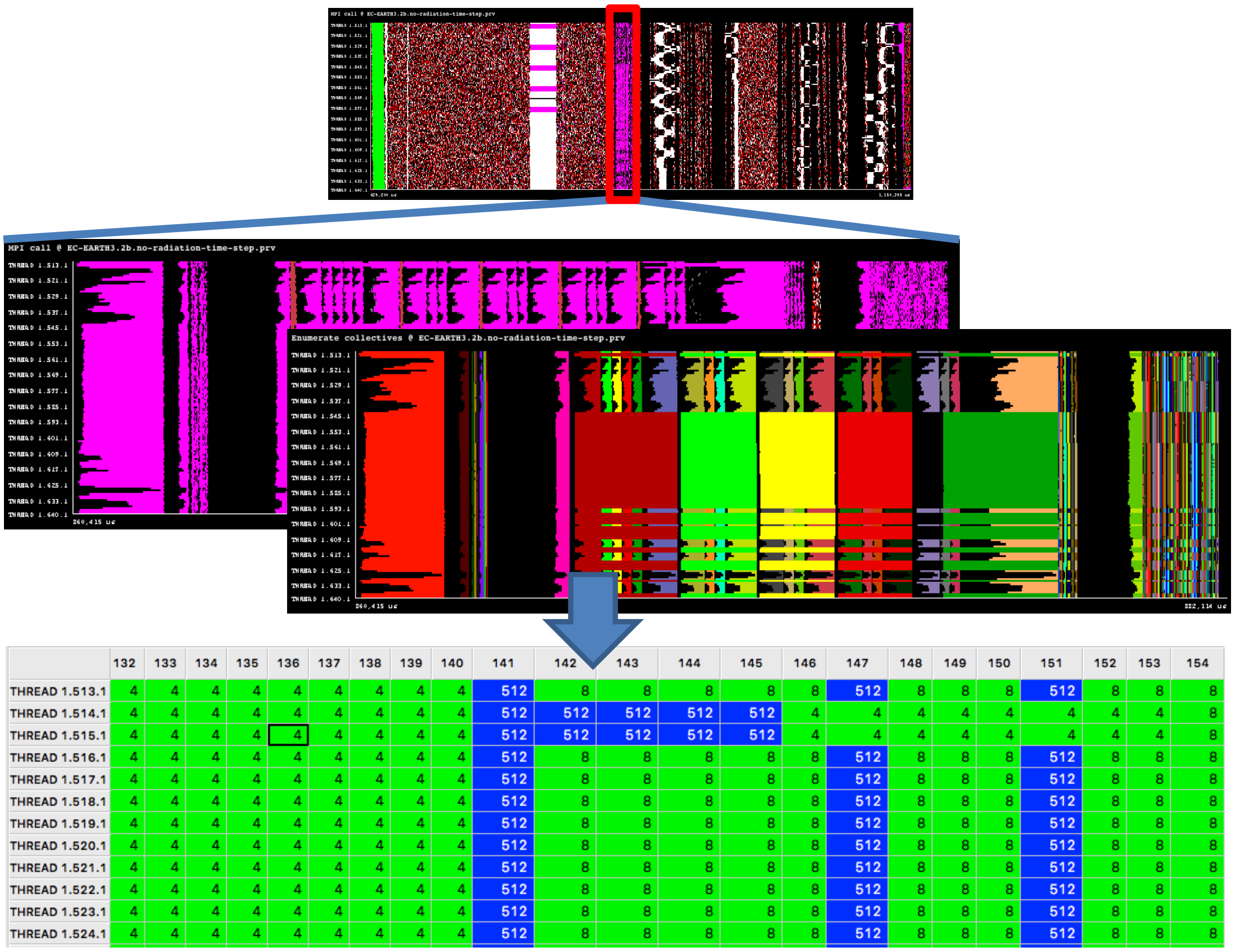
Figure 35: One NEMO process suffers from preemption and as a consequence the rest of processes have to wait. It is visible in the useful duration trace that it has a large computation, but very slow, having only 0.04 of IPC.

[4] One possible solution could be to modify and re-compile the operating system's kernel to change the process scheduling, but this could lead to instabilities. Moreover, it is not feasible in real supercomputing environments.

4.3.2.2.3. Global communications

The third remarkable issue is related to some global communications done through several MPI_Allreduce calls. Figure 36 shows the size of some of these calls. There is also a trace used to show all the different collective MPI calls.

The main issue is that there is a large number of communications and each communication is small (4, 8 and 512 bytes). This implies to be penalized by network latencies. Furthermore, the bandwidth is underused.


MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
Enumerate collectives @ EC-EARTH3.2b.no-radiation-time-step.prv
860,415 us
882,114 us

	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154
THREAD 1.513.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.514.1	4	4	4	4	4	4	4	4	4	512	512	512	512	512	4	4	4	4	4	4	4	4	8
THREAD 1.515.1	4	4	4	4	4	4	4	4	4	512	512	512	512	512	4	4	4	4	4	4	4	4	8
THREAD 1.516.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.517.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.518.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.519.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.520.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.521.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.522.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.523.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8
THREAD 1.524.1	4	4	4	4	4	4	4	4	4	512	8	8	8	8	8	512	8	8	8	512	8	8	8

Figure 36: There are many MPI_Allreduce calls in a row, all them painted with a trace. Moreover, it is shown the size of some of them, which have small sizes: 4, 8 and 512 bytes.

4.3.2.2.4. Workload imbalances

The final issue to be considered is the load imbalance focused on some regions of NEMO. In Figure 37 there is an example of one of these regions. The histogram of correlated instructions and IPC demonstrates that there is a lot of variability in the number of instructions to execute, i.e., the workload.

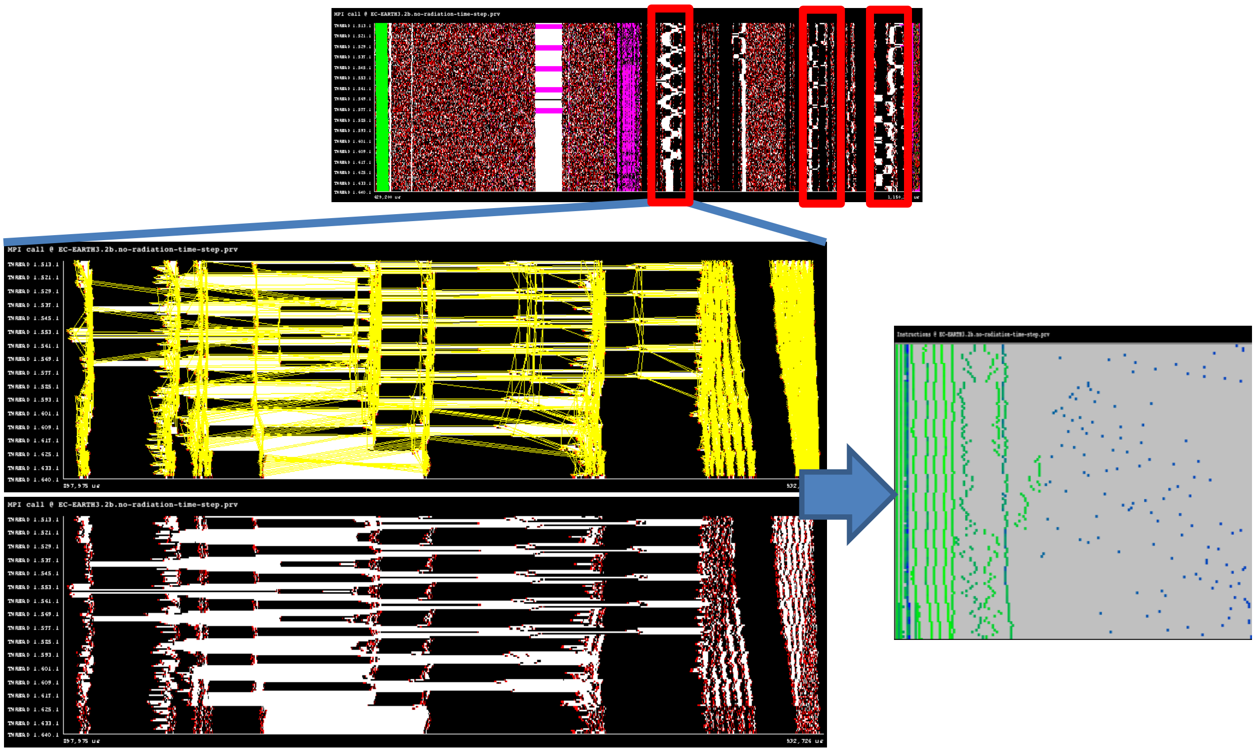

MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
MPI call @ EC-EARTH3.2b.no-radiation-time-step.prv
Instructions @ EC-EARTH3.2b.no-radiation-time-step.prv

Figure 37: Example of one of the imbalances. The histogram on the right is showing the correlated instructions with the IPC and it is deducible that the workload balance is not good. There is a lot of variability, as each NEMO process executes a different amount of instructions but having the same IPC.

5.Conclusions

The execution of EC-Earth needs a lot of resources and they are not unlimited, so it is necessary to use them as efficiently as possible. For this reason, in this study we have presented a scalability analysis in order to know how the release 3.2.0 of EC-Earth scales on MareNostrum III, which are the MPI combinations with the best performance-efficiency compromise and which are the combinations that achieve a desired throughput expressed in SYPD. Additionally, this study also includes another part that consists in doing a performance analysis of EC-Earth. We have included both profiling and tracing analyses, identifying where are the bottlenecks that affect considerably the parallel efficiency.

To achieve this, it has been necessary to develop a methodology to properly study a bi-dimensional scalability, as we are doing the scalability analysis using two parallel models coupled, IFS and NEMO, and for both we suppose that it is necessary to set different amounts of MPI processes. This has an additional problem, which is the representation of 3D data in an understandable way, simplifying the search in order to use 2D data. For doing this, it is necessary to find a way to reduce the amount of combinations, but keeping the most relevant ones. To solve it, we have presented a new metric called performance-efficiency compromise which is calculated as the product of the speedup and the efficiency. In the end, we have got 2D charts that easily show how EC-Earth behaves.

We have seen that EC-Earth does not scale well, as the efficiency is really low. However, we have pointed out which are the most interesting MPI combinations to be used, depending on the needs of the user, i.e., if the performance of the model is more important for a particular study, if it is preferable a good balance between performance and efficiency, etc. In particular, the fastest combination is 640, using 512 MPI processes for IFS and 128 for NEMO. It has a speedup of 40.3 and gives a throughput of 15.7 SYPD. On the other hand, using the new performance-efficiency compromise metric, we see that the best combination is using 416 MPI processes, 288 for IFS and 128 for NEMO. It has a speedup of 35 and is able to achieve about 13.6 SYPD.

On the other hand, we also present a complete performance analysis to identify where are the main bottlenecks of EC-Earth, focusing on the coupling between IFS and NEMO through OASIS3-MCT. We have seen that performing a profiling analysis with gprof is not enough, so it has been necessary to use more powerful tools such as Extrae and Paraver to trace the model.

We have realized that the main issue of the version 3.2.0 of EC-Earth is the coupling of variables from IFS to NEMO, because some fields should be sent in a conservative way and the algorithm applied to them is very expensive in terms of time for parallel executions, as it serializes a large section of the IFS time step, reducing the parallel efficiency of the model drastically.

Furthermore, IFS has a workload imbalance due to Semi-Lagrangian treatment of advection in physical space. In computational terms, this implies that processors assigned to the equator longitudes have less work than processors assigned to the pole areas.

In NEMO, the major issue is with regard to the granularity of its computational bursts. They are too small because the tendency of NEMO is to calculate something and communicate it, which decreases the efficiency considerably due to the latency and synchronization introduced as MPI overhead. It would be more desirable to have larger computational bursts to reduce this overhead due to communications, and package these communications whenever possible, in order to reduce the latency and use correctly the bandwidth of the network.

In addition, NEMO also has some minor issues that have an impact on performance. We have explained that preemption, which is related to the operating system, is not feasible to fix. However, we have identified other 3 problems that could be worth to optimize. The first one is the synchronization of NEMO that waits for runoff-mapper's data. The second issue is a series of global communications in a row using MPI_Allreduce. And the last one are some workload imbalances distributed along the NEMO time step.

Given that EC-Earth will contribute to the next Coupled Model Intercomparison Project Phase 6 (CMIP6), it is essential to improve its efficiency because it will be needed to perform an enormous amount of experiments. Since all these experiments require a large number of computing hours and resources, every small performance improvement of the model would increase the energy efficiency of future experiments.

Acknowledgements

The research leading to these results has received funding from the EU ESiWACE H2020 Framework Programme under grant agreement n° 675191.

References

[1] Asif M., A. Cencerrado, O. Mula-Valls, D. Manubens, A. Cortés and F.J. Doblas-Reyes, 2014: “Case Study in Large Scale Climate Simulations: Optimizing the Speedup/Efficiency Balance in Supercomputing Environments”. 14th International Conference on Computational Science and Its Applications, doi:10.1109/ICCSA.2014.57

[2] Hazeleger W., X. Wang, C. Severijns, S. Ştefănescu, R. Bintanja, A. Sterl, K. Wyser, T. Semmler, S. Yang, B. van den Hurk, T. van Noije, E. van der Linden, K. van der Wiel, 2011: “EC-Earth V2.2: description and validation of a new seamless earth system prediction model”. Clim Dyn, doi:10.1007/s00382-011-1228-5